Resumen

Los widgets, pequeñas aplicaciones de escritorio, se han convertido en objeto de estudio para este trabajo. Se ha analizado y estudiado su entorno, funcionamiento y estructura. Como resultado se ha obtenido una definición del concepto de widget, con diferentes ejemplos prácticos, y una comparativa entre las compañías más destacadas que ofrecen widgets (Apple, Microsoft, Yahoo!, Google y Opera) así como de las diferencias a nivel de estructura e implementación que se encuentran entre sus widgets.

En el apartado de estudio también se ha dado atención a los usuarios de estas aplicaciones, explicando cuáles son sus necesidades, costumbres y posibilidades de elección en este campo. Otro de los apartados trata de cubrir algunas de las tecnologías más relevantes implicadas en la creación posterior de un widget, la aplicación Dashcode, el lenguaje de programación JavaScript, y los canales RSS.

El tema de los widgets también ha dado lugar a todo un apartado de desarrollo. Se ha creado un widget para la plataforma de Apple, Macintosh. El widget proporciona al usuario las últimas noticias y resultados del mundo deportivo, concretamente de dos deportes, fútbol y baloncesto.

Todas las fases del proceso de creación del widget han sido descritas de principio a fin, desde los bocetos iniciales a la puesta en funcionamiento, pasando por el prototipado e implementación. De manera adicional, también se ha incluido un apartado dedicado al diseño de widgets, con algunas y consejos a tener en cuenta al diseñar widgets en general.

Por último se han vertido diferentes conclusiones fruto del presente trabajo y de la reflexión sobre la situación actual de este campo, sin olvidar los aspectos a mejorar, las áreas en que se está trabajando y las perspectivas de futuro.

Tabla de Contenidos

Índice de Tablas y Figuras

I - Tablas

I - BLOQUE ESTADO DEL ARTE

V - ANEXO

II - Figuras

I - BLOQUE ESTADO DEL ARTE

modificar los atributos del objeto seleccionado, y justo debajo la ventana “Library”, con diferentes elementos que se pueden añadir al widget. En la parte inferior, un editor de texto para visualizar o modificar los archivos de código

Figura 1.22: Pantalla de selección de plantillas para iniciar un proyecto con Dashcode

Figura 1.23: Organización jerárquica del DOM

Figura 1.24: Icono que denota la presencia de un canal RSS. Se ha convertido en un estándar de facto

II - BLOQUE DE DESARROLLO

Figura 2.1: Boceto de un widget que permitiría subir archivos al servidor www.megaupload.com directamente, sin entrar en la página, retornando un enlace al archivo subido

Figura 2.2: Boceto de un widget para consultar los dominios asociados a direcciones IP dinámicas

Figura 2.3: Boceto de un widget para mostrar al usuario la ruta de carretera a seguir para alcanzar al destino introducido, desde el origen deseado

Figura 2.4: Boceto de un widget para localizar los paquetes de correos a partir de su código de paquete

Figura 2.5: Boceto del juego de 3 en raya en forma de widget, con la salvedad de que pretendía ofrecer juego online, partidas contra otros jugadores a través de internet

Figura 2.6: Boceto de un widget para descargar vídeos de diferentes servidores (youtube.com, video.google.com, etc;) en la máquina del usuario

Figura 2.7: Boceto de un widget que pretendía facilitar la tarea de subir vídeos a www.youtube.com permitiendo arrastrar nuestros vídeos sobre el widget y obtener directamente un link al mismo una vez subido

Figura 2.8: Boceto de un widget para consultar rápidamente las últimas noticias y resultados del mundo del deporte, acabó siendo el candidato ganador.

Figura 2.9: Anverso del widget en la primera fase de prototipado

Figura 2.10: Prototipo de los anversos del widget en cada modo diferente

Objetivos

De reciente aparición son los Widgets, también conocidos como Gadgets, pequeñas aplicaciones que parecen haber brotado en nuestros escritorios de un tiempo a esta parte. A pesar de que su uso se ha empezado a generalizar de forma lenta pero inexorable, parece lógico preguntarse por qué han llegado a formar parte de nuestros escritorios y cómo funcionan realmente. Así pues, el presente trabajo tiene el propósito de cumplir los siguientes objetivos:

- Definir de manera formal el término "widget" y justificar su uso y aparición en el mercado. No todo el mundo está familiarizado con estas aplicaciones y todavía existe mucho desconocimiento al respecto, por lo que resultará interesante profundizar en los orígenes y significado de las mismas.

- Analizar y comparar de forma objetiva las diferentes posibilidades que existen en el mercado tanto para utilizar widgets a nivel de usuario como para desarrollarlos. También determinar qué tecnologías son necesarias y cuáles son sus ventajas y desventajas. En muchas ocasiones, los usuarios y desarrolladores de widgets no son conscientes de que existen alternativas a los widgets que han estado utilizando, quizás porque su equipo los incluía por defecto. Por ello se tratará de profundizar en las alternativas reales y factibles en lo que a widgets se refiere.

- Implementar un widget concreto para la plataforma Macintosh, facilitando la comprensión tanto de la estructura interna de un widget como de su aspecto exterior y de las tecnologías implicadas en el proceso. Tras haber explicado sobre el papel qué es un widget y cómo está hecho, parece lógico adentrarse de lleno en el proceso práctico de creación y comprobar realmente las posibilidades que ofrecen. Diseñar e implementar un widget es la mejor manera de comprender su manejo y funcionamiento.

Introducción

El punto de partida de este trabajo fue la plataforma Macintosh. Como usuario de Mac, tenía claro que quería realizar un proyecto que girase en torno a este sistema quizás no muy conocido, y no demasiado popular, pero ciertamente con unas posibilidades inmensas y unas características poco menos que asombrosas. El hecho de ser un terreno relativamente poco explorado en el ámbito de la investigación también supuso un punto a favor de la decisión de trabajar con esta plataforma. Además, Mac OS X incluye XCode, un completo entorno de trabajo para desarrolladores que, además de gestionar los diferentes proyectos, proporciona soporte y documentación sobre las tecnologías envueltas en Mac OS X.

Tras barajar diferentes posibilidades, empezó a ganar peso la idea de realizar un trabajo en torno al mundo de los widgets. Los widgets son parte integral de Mac OS X Tiger, y se presentan como una de sus características destacadas a través de Dashboard (la aplicación que los gestiona), pero no es la única plataforma que los soporta, ni mucho menos. Como usuario habitual de widgets, parecía una buena idea profundizar en el tema, ver qué alternativas existen, cómo funcionan los widgets, las tecnologías envueltas y cómo se implementan, lo que supuso el inicio del presente trabajo.

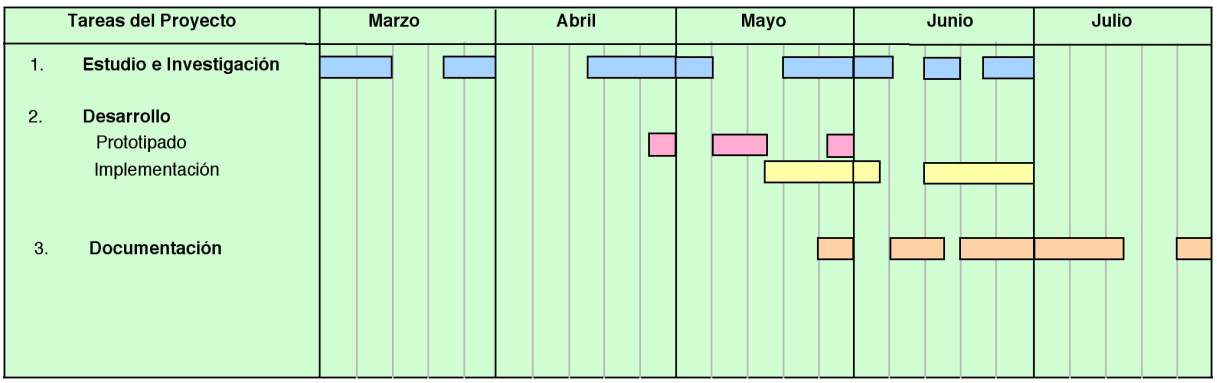
Por consiguiente, a lo largo de este documento veremos qué es un widget, de dónde surgen, cómo y dónde se pueden utilizar, etc. Se analizarán las tecnologías envueltas en su funcionamiento y desarrollo y también se detallará el proceso de creación de un widget para consultar las últimas noticias y resultados del mundo del deporte. Todo ello con la finalidad de proporcionar una visión general bastante completa en torno al mundo de los widgets.

Temporización

El proyecto empezó a tomar formar a principios del 2007. Antes de dar con la idea de los widgets, barajamos la posibilidad de hacer el trabajo sobre otro tema, también basado en Mac (WebObjects), del cual no se llegó a hacer una propuesta. Finalmente nos decantamos por el tema de los widgets.

La fecha prevista para acabar el proyecto era Junio - 2007. La realidad ha resultado ser distinta, puesto que el proyecto ha sufrido un retardo en este respecto. Los primeros hitos marcados, de hecho, se ajustaron bastante a la temporalidad prevista inicialmente. Más adelante, se empezaron a producir retardos en algunas de las etapas. Quizás fue un poco ambicioso fijar la fecha límite para tal mes, en vista del trabajo envuelto, aunque también es cierto que diferentes circunstancias de carácter personal (inicio de un trabajo seglar, entre otros) han contribuido en el retardo de la entrega.

En cualquier caso, el número de horas invertidas se ajusta bastante al número de créditos asignados al trabajo. La parte que más dedicación ha requerido ha sido la de documentación e investigación, seguida por la implementación, que se ha aligerado gracias al uso de herramientas como Dashcode (detallada más adelante). El esfuerzo realizado se puede observar en el siguiente cronograma de actividades:



El desglose en número de horas es el siguiente:

- Estudio e Investigación: 48,5 horas
- Implementación: 38 horas
- Documentación: 64,5 horas

Total: 151 horas

El presente trabajo no tiene como único objetivo la implementación y desarrollo de una aplicación (un widget en este caso). Por consiguiente, los datos que se obtienen de un presupuesto no son del todo representativos de la magnitud y coste del proyecto. Tampoco resultan orientativos en este caso otros datos como las línea de código empleadas. En cualquier caso, proporcionaré un presupuesto orientativo del widget en cuestión, teniendo en cuenta el tiempo invertido en el aprendizaje de las tecnologías empleadas y los estudios realizados.

Asumiendo un salario medio de 15€ / hora de un programador, tendremos en cuenta que se han dedicado 33 horas a la implementación, y repercutiremos un 15 % del tiempo invertido tanto en el aprendizaje de las tecnologías como en el estudio y análisis de los widgets y elaboración del documento final. El cálculo es el siguiente:

15€/horas * (33 horas implementación + 0,15 * 42,5 horas de investigación + 0,15 * 54,5 horas de documentación) = 713,02 €

Estructura del Documento

El presente trabajo está dividido en cuatro bloques diferentes y un anexo: bloque introductor, bloque de estado del arte, bloque de desarrollo y bloque de finalización.

El bloque introductor se centra en presentar los contenidos y el origen del trabajo, así como los objetivos que se pretenden conseguir. También detalla la temporización del proyecto y su coste e incluye los correspondientes índices de tablas, figuras y contenidos.

Por otra parte, el bloque de estado del arte conforma un elevado porcentaje del trabajo correspondientes al estudio e investigación de los widgets y su entorno. Encontraremos por tanto en esta sección la definición del término "widget", un análisis de las diferentes compañías que los ofrecen y las principales diferencias entre ellas en lo referente al uso y creación de los widgets. Se incluye también una descripción de las tecnologías más destacables envueltas en el diseño y uso de los widgets.

El tercer bloque, el bloque de desarrollo, cubre todo el proceso de creación de un widget para plataforma Macintosh, desde los bocetos iniciales a la publicación final del widget. También se detallan algunas pautas de diseño a tener en cuenta durante la implementación de los widgets en general.

Por último, en el bloque de finalización aparecen las conclusiones obtenidas del trabajo, en base a los objetivos iniciales, así como conclusiones más específicas de carácter más personal. Otras de las secciones de este bloque es el apartado "Trabajo Futuro", donde se explica qué apartados del trabajo se podrían mejorar así como otras ideas que podrían haberse incluido. Se hace hincapié también en los progresos a realizar en el mundo de los widgets y la dirección que están tomando las diferentes compañías envueltas.

En el anexo se puede encontrar información más detallada sobre algunas porciones del código utilizadas, incluyendo las clases y objetos predefinidos.

I - BLOQUE ESTADO DEL ARTE

¿Qué es un widget?

Definición y concepto

El término original "widget" procede del inglés y hace referencia a un dispositivo mecánico, especialmente aquellos cuyo nombre es desconocido o no ha sido especificado. En el campo de la informática, se ha acuñado la palabra "widget" para describir un componente informático que forma parte de una interfaz gráfica. Sin embargo, el término, tal y como se utiliza en el presente documento tiene un significado mucho más concreto.

Entendemos un "widget" como una aplicación de tamaño reducido y con funcionalidad muy limitada y concreta, cuya finalidad es proporcionar al usuario un determinado servicio de forma rápida y sencilla. Generalmente ofrecen funciones de consulta u otras operaciones, pero siempre de carácter breve, permitiendo un fácil acceso a las mismas. Los widgets son ejecutados por un servidor o motor de widgets en la máquina del usuario, y suelen ocupar una ubicación de fácil acceso en el escritorio. También existen widgets que necesitan de una conexión a internet para funcionar, puesto que se ejecutan a través de un servidor externo, mostrándose en el escritorio del terminal.

Un ejemplo de widget muy concreto sería el de un reloj.

Tal y como se puede observar en la figura 1, captura de un widget a tamaño real, el área ocupada suele ser bastante reducida, y el diseño simple y agradable a la vista.

También podemos destacar de la figura 1 el pequeño botón con la letra 'i' que aparece en la esquina inferior derecha. Este botón, presente en la mayoría de widgets, permite acceder al reverso de los widgets (Figura 1.1). Es una manera de ampliar las posibilidades de interacción y funcionalidad de los widgets ocupando el mínimo espacio.

A pesar de que desde el punto de vista del usuario los widgets son vistos como aplicaciones muy simples, en realidad están potenciados por tecnologías web y estándares tales como HTML, CSS, y JavaScript. Como veremos más adelantes, cada compañía añade además otras particularidades que extienden las posibilidades de los widgets.

Figura 1.1: Parte frontal del widget Reloj Internacional

Figura 1.2: Reverso del widget Reloj Internacional

El origen de los Widgets

El concepto de widget tal y como se entiende en el presente documento tiene unos orígenes más bien borrosos, y existe cierto debate sobre quién realmente invento estas pequeñas aplicaciones.

En 1984 Apple reveló su sistema operativo Mac OS, el cual incluía, entre otras características una función que denominaron "Desk Accesories", accesorios de escritorio, fácilmente comparables con Dashboard, la aplicación de widgets de su último sistema operativo. Es interesante observar que esta función no se incluyó en las primeras versiones de Mac OS X. Mientras tanto, una pequeña compañía llamada Pixoria lanzó al mercado "Konfabulator", un entorno que permitía ejecutar pequeñas aplicaciones escritas en JavaScript en el escritorio. Konfabulator supuso una clara mejora sobre el concepto introducido por "Desk Accesories", pero no se sabe a ciencia cierta si fueron estos los que motivaron su creación. Un poco más adelante, Apple lanzó al mercado Dashboard, como parte integral de Mac OS X 10.4, Tiger y Pixoria fue comprada por Yahoo!, que amplió el uso de Konfabulator y lo puso a la libre disposición de los usuarios.

Entones, ¿quién inventó los widgets? Aunque los "Desk Accesories" de Apple fueron la primera alternativa disponible en el mercado, lo cierto es que los widgets de Dashboard se parecen mucho más a la aplicación "Konfabulator" de Pixoria que a los propios "Desk Accesories". En cualquier caso, la aparición de los widgets siempre fue motivada como una respuesta a la necesidad poder disponer de un recurso necesario de forma rápida y sencilla, sin tener que hacer uso de aplicaciones multi-funcionales.

Ejemplos de uso

Los widgets se pueden clasificar en diferentes categorías según la temática que comprenden o funcionalidad que proporcionan. Así pues diferenciamos entre los siguientes tipos de widget:

- **Información**

  Como su propio nombre indica, estos widgets tienen como objetivo proporcionar al usuario datos o información necesaria de forma rápida y sencilla. Los ejemplos más comunes en este apartado serían los relojes, o widgets de información meteorológica, por mencionar algunos.

  En la figura 1.3, una captura del widget del tiempo que aparece por defecto en el Dashboard de Apple.

Figura 1.3: Widget Weather de Apple

- **Cálculos y conversiones**

  Están destinados a realizar cálculos o conversiones concretas. El catálogo en este apartado es muy variado, desde simples calculadoras a calculadoras científicas, pasando por conversores de divisas.

Figura 1.4: Widget Calculadora

- **Noticias**

  Estos widgets muestran al usuario las últimas noticias de actualidad, obtenidas a través de una conexión a internet de una o varias páginas web. Dentro de esta misma categoría la temática puede ser muy variada, encontraremos widgets que obtienen las últimas noticias que aparecen en blogs o foros, o las relacionadas con un campo concreto de investigación o entretenimiento. El widget se encarga de acceder a la página, recoger la información deseada y mostrarla al usuario, facilitando la tarea de consulta.

Figura 1.5: Widget de noticias

- **Status**

  Widgets muy comunes cuya función generalmente es proporcionar al usuario información sobre el estado de su equipo: duración de la batería (en caso de portátiles), uso de CPU, detección de redes inalámbricas, etc.

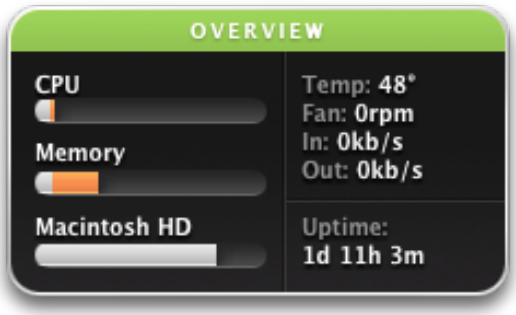


Figura 1.6: Widget iStat Nano

- **Juegos**

  El objetivo en este caso es entretener al usuario, de forma rápida y sin complicaciones. En este apartado encontramos también una amplia variedad de widgets: Sudokus, crucigramas, puzzles, arcades, etc;

Figura 1.7: El juego de los ladrillos

- **Películas, Radio y Televisión**

  Otra categoría correspondiente al mundo del entretenimiento. En esta ocasión se pretende proporcionar la posibilidad de consultar los últimos estrenos de cine, escuchar sus emisoras de radio preferidas o incluso ver su programa de televisión favorito.

  La mayoría de estos widgets requieren de una conexión a internet para su correcto funcionamiento.

Figura 1.8: Widget BBC Radio

Figura 1.9: Widget Diccionario Inglés

- **Referencia**

  Permiten al usuario acceder a diferentes obras o servicios de consulta, tales como diccionarios (en diferentes idiomas), tutoriales de múltiples aplicaciones o lenguajes de programación, y un largo etc.

- **Búsqueda**

  Estos widgets proporcionan servicios de búsqueda de todo tipo, desde búsqueda de páginas en Internet, hasta búsquedas en el sistema de archivos del usuario. También los hay destinados a buscar productos muy particulares: restaurantes, gasolineras, etc.

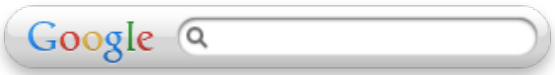


Figura 1.10: Widget buscador Google

- **Transporte y Viajes**

  Permiten al usuario monitorizar vuelos, trenes y otros servicios de transporte, así como obtener información sobre diferentes países turísticos.

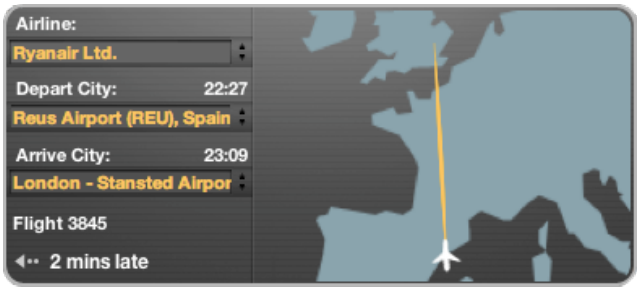


Figura 1.11: Flight Tracker, seguidor de vuelos

- **Desarrollo**

  Todo tipo de herramientas que puedan resultar de utilidad para desarrolladores: índices de comandos, compiladores, etc. La mayoría de lenguajes de programación disponen de widgets que pueden resultar de utilidad para el programador.

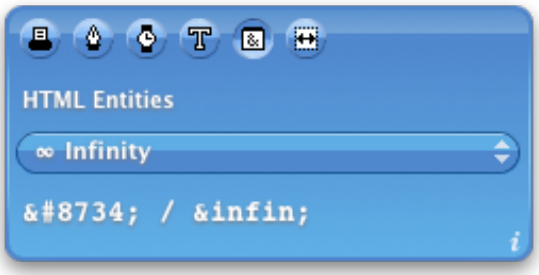


Figura 1.12: Widget para desarrolladores

El espectro de widgets que se encuentra en el mercado es muy amplio, en este apartado se ha cubierto una pequeña muestra de los más populares. La relativa facilidad de uso de las tecnologías sobre las que se basan, así como la popularidad de las mismas ha impulsado su uso y creación.

Diferentes compañías que ofrecen widgets

Actualmente se encuentran en el mercado cinco compañías mayoritarias que ofrecen entre sus servicios el uso y creación de widgets. Son las siguientes:

1.*Apple Inc.*

Apple es la principal productora de una serie de ordenadores personales bajo el nombre comercial Apple Macintosh, así como de diferentes periféricos y software. Además, a día de hoy lidera el mercado de música digital.

2.*Yahoo! Inc.* Compañía líder en comunicaciones y comercio que ofrece una amplia red de servicios a millones de usuario alrededor del mundo. También proporciona negocios online y servicios de empresa destinados a aumentar la productividad de sus clientes.

3.*Google Inc.* Es una empresa especializada en búsquedas en Internet y publicidad online. Su principal producto es un potente motor de búsqueda con el mismo nombre, aunque también ofrece muchos otros servicios como correo electrónico (gmail) o mensajería instantánea.

4.*Microsoft Corporation.* Multinacional corporativa que desarrolla, produce y da soporte a una amplia variedad de software para ordenadores. Sus productos estrella son el sistema operativo Windows y la herramienta de trabajo Microsoft Office.

5.*Opera Software.* Corporación conocida principalmente por sus aplicaciones Internet, siendo el navegador Opera la más destacable. Opera Software también está envuelta en la promoción de los estándares web, colaborando con el W3C.

Diseño e Implementación en las Diferentes Plataformas

Veamos cómo aborda cada una de las compañías la tarea de creación y desarrollo de estas pequeñas aplicaciones, así como las herramientas y facilidades que ofrecen al usuario con tal finalidad.

Apple

Introdujo los widgets por primera vez en Tiger, uno de sus últimos sistemas operativos (basado en MAC OS X), a través de una aplicación llamada "Dashboard", un servidor que se encarga de gestionar y presentar las mini-aplicaciones. Se profundizará más acerca de Dashboard en la sección correspondiente.

Estos widgets están basados en tecnologías Web Kit, y se crean combinando HTML, JavaScript y CSS, lo cual extiende la capacidad de desarrollar widgets a un amplio público dada la popularidad de tales herramientas. Su uso, sin embargo, no está limitado a tecnologías web de común utilización, sino que también hacen uso de las cualidades únicas que proporciona MAC OS X, como son los comandos UNIX, Quartz Drawing (visualización de gráficos), Plug-ins de Internet o Plug-ins para widgets.

Básicamente estos widgets son una página web sencilla que se muestra en el Dashboard en lugar de en el navegador habitual. Están almacenados en el disco en un paquete que agrupa todos los recursos necesarios en el mismo lugar. Un widget básico contiene los siguientes archivos:

- Un archivo HTML que define la interfaz de usuario del widget
- Una imagen de fondo por defecto con extensión PNG que se muestra en el Dashboard mientras se carga el widget.
- Un icono, también en formato PNG usado para representar el widget en la barra de widgets
- Un archivo de propiedades llamado `Info.plist` que contiene un identificador de widget, el nombre, información de la versión, tamaño, etc.

A medida que el widget va creciendo en complejidad, se pueden añadir otros archivos, como imágenes o archivos CSS y JavaScript externos, dentro del mismo paquete.

Finalmente, para crear el widget basta con crear un directorio donde almacenar los archivos antecitados, y añadir a la carpeta la extensión `.wdgt`, el sistema reconoce el archivo y el widget ya está listo para ser usado. Al hacer doble clic, se instala y ya aparece en la barra de widgets.

YAHOO!

Esta compañía realiza la creación y gestión de widgets a través de lo que se conoce como "The Yahoo! widget Engine" (anteriormente denominado "Konfabulator"), un motor que ejecuta JavaScript en tiempo de ejecución, disponible para Windows y Mac OS X. Este motor o aplicación es la que permite ejecutar los widgets en el terminal del usuario.

Los widgets que conforman este sistema basan su funcionamiento, básicamente, en XML y JavaScript, dos tecnologías muy populares que se utilizan para muchos otros fines. Lo que diferencia estos widgets, es que hacen uso de una API específica del motor de widgets de Yahoo!, permitiendo definir funcionalidad adicional.

Estos widgets se presentan en 3 diferentes formatos, de los cuáles sólo 2 se utilizan actualmente. El más común es ZIP (en esencia un archivo ZIP renombrado con extensión `.widget`), aunque los widgets "planos" (sin compresión) están ganando popular, puesto que se arrancan más rápido y permiten firmar el paquete digitalmente, verificando la identidad del autor.

En el paquete de un widget podemos encontrar los siguientes archivos:

- `.kon`

Contiene el código principal del widget. El motor de widget lee este archivo en primer lugar cuando el usuario hace doble clic sobre el paquete. Generalmente contiene información referente a la posición inicial y referencias (XML) de la imagen de portada, así como otras preferencias. Suele contener la mayoría del código que define la funcionalidad del widget (JavaScript), aunque esto no siempre es así. En ocasiones, el código JavaScript se almacena en un archivo .js aparte, sobre todo en widgets complejos.

- `.js`

Si está presente, contiene la mayor parte del código JavaScript que el widget necesita para funcionar. Este archivo no contiene XML.

- `.Info.plist`

Otro archivo que utiliza XML, accedido por Mac OS X para obtener información sobre la versión del widget que se va a mostrar. Windows ignora este archivo, así como Mac OS X, en caso de que el formato del paquete del widget sea multiplataforma.

- `.scpt`

Es un documento que contiene comandos AppleScript. Este archivo suele aparece en algunos casos para reducir la complejidad de los archivos .js o .kon. Un widget que contiene este archivo sólo puede ser ejecutado en un Mac.

También pueden aparecer otros tipos de archivos, como imágenes o sonidos, aunque los antecitados son los más habituales.

MICROSOFT

Introduce su particular versión de widgets en su último sistema operativo: Windows Vista, donde estas pequeñas aplicaciones se conocen como "Gadgets". Su uso y manipulación se realizan a través de una barra lateral (Sidebar), aplicación que permite al usuario mostrar los "Gadgets" en el escritorio de windows y en la propia barra lateral.

Los Gadgets se desarrollan utilizando HTML y técnicas de scripting, y contienen los siguientes archivos:

- Manifiesto Gadget: Es un archivo XML que define las propiedades del gadget, incluyendo el nombre, icono y descripción
- Archivo HTML: Define el código principal del gadget
- Scripts (JavaScript): Determina la lógica del Gadget
- Imágenes, hojas de estilo (CSS): Complementan el código principal, enriqueciendo la interfaz de usuario
- Icono: Visión en miniatura del Gadget para seleccionarlo en la galería

Adicionalmente, se puede ampliar la funcionalidad de nuestros Gadgets haciendo uso de las APIs (Application Programming Interface) que proporciona la plataforma de Gadgets para interaccionar con el Sistema Operativo (lectura y escritura de ficheros, mostrar información sobre Hardware, redes, etc.)

La plataforma de Gadgets también ofrece soporte para localización, permitiendo mostrar el Gadget en el idioma preferido por el usuario, siempre y cuando el creador de la aplicación haya realizado los ajustes necesarios. Es necesario crear una subcarpeta que coincida con el identificador local de Windows para cada idioma diferente, donde se incluirían los archivos traducidos. Los archivos comunes, que no requieren de localización (ciertas imágenes, algunos scripts, etc.), no tienen la necesidad de repetirse, pueden aparecer una única vez en el directorio principal.

Cuando hemos reunido todos los archivos necesarios para el funcionamiento del Gadget en una misma carpeta contenedora, basta con añadirle la extensión .gadget para que el propio sistema lo reconozca como un Gadget ejecutable en la barra lateral.

Existe también la posibilidad de usar técnicas de compresión varias, añadiendo la extensión .ZIP o .CAB (Windows Cabinet File), aunque deberán ir siempre acompañados por la extensión .gadget para que Windows pueda reconocer el archivo una vez descomprimido.

GOOGLE

Los widgets de Google se denominan "Google Gadgets", y se clasifican en dos tipos diferentes: Universal Gadgets y Desktop Gadgets.

- Universal Gadgets

Pueden funcionar en páginas web del usuario, así como en múltiples aplicaciones o servicios de Google (Google Homepage, Google Desktop, Google Page Creator, Blogger). Sin embargo, están sujetos a las limitaciones del entorno del navegador (elemento imprescindible) y sólo funcionarán si el usuario está conectado a Internet.

- Desktop Gadgets

Funcionan exclusivamente con Google Desktop. No necesita un navegador web abierto, se ejecutan en el escritorio del usuario y no hace falta conexión a Internet. Es necesario descargar e instalar un kit de desarrollo para su correcto funcionamiento (SDK). Soporta lenguajes de programación adicionales, a parte de JavaScript, como C, C++, C# y VB.net.

A pesar de las diferencias básicas de funcionamiento entre los dos tipos de widget, comparten los mismos pilares de diseño e implementación. Google proporciona una API específica para la creación de Gadgets: Google Gadgets API.

La API de Google Gadgets consiste en una serie de bloques de construcción: XML, HTML y JavaScript, los cuales se detallan a continuación:

- XML: Fichero que se encuentra en un servidor de Internet que contiene las instrucciones de procesamiento del Gadget. Puede contener todos los datos y código del gadget o bien puede tener referencias (URL's) donde encontrar el resto de elementos. Conocido como "el manifiesto del Gadget".
- HTML: Presenta el contenido estático del gadget, y también permite incrustar código css para dar formato al widget.
- JavaScript: Permite incorporar comportamiento dinámico.

Cuando tenemos listos los contenidos del gadget, hay que cargarlos en un servidor (Google proporciona el servicio gratuitamente) en el caso de que estemos desarrollando un Gadget universal.

A diferencia de los widgets de otras compañías, en esta ocasión, no hay que colocar todos los ficheros en una misma carpeta, sino que todo el contenido del Gadget aparece en el archivo .xml, el cual, en caso necesario, contendrá los enlaces al resto de archivos, que también estarán en la red. En el caso de los Gadgets de escritorio, se debe proceder a la descarga de los archivos correspondientes, y se ejecutan mediante la aplicación Google Desktop.

Opera

Ofrece a sus usuarios la posibilidad de usar widgets a través de su popular navegador Web, Opera Browser. Es el propio navegador quien se encarga de instalar y abrir los widgets, y mostrarlos directamente en el escritorio del usuario.

Un widget de Opera es un archivo zip corriente renombrado con la extensión .wdgt. Todos los archivos necesarios para el correcto funcionamiento del mismo se encuentran dentro del archivo. Estos widgets suelen contener los siguientes elementos:

- Un archivo de configuración del widget, que contiene información relevante para su uso (tamaño, nombre, autor, etc.), de extensión XML
- Documento HTML. Es el esqueleto del widget. También se pueden crear usando contenidos que el navegador Opera maneja de forma nativa como archivos SVG o XML.
- Imágenes
- Archivos JavaScript
- Hojas de Estilo

Cuando hemos creado el widget, podemos publicarlo a través de la comunidad Opera (My Opera Community), y mostrarlo al resto de usuarios.

De nuevo, estos widgets son muy similares a las páginas web, aunque tienen algunas diferencias:

Los widgets son externos al navegador, se pueden colocar en el escritorio, aunque es el propio navegador el encargado de ejecutarlos.

Cada widget tiene acceso a un espacio de almacenamiento permanente donde se guardan sus opciones de configuración, así como cualquier información descargada. El mecanismo es similar al de las cookies, con la diferencia de que la capacidad de almacenamiento es mayor y no se elimina automáticamente tras un periodo de tiempo.

Opera diferencia sus widgets del resto ofreciendo sus propias librerías para dotarlos de funcionalidad extra, a parte de la que proporcionan las tecnologías base. La más común y utilizada es la librería de animación de Opera (Opera Animation Library).

Principales diferencias

Una vez explicadas las características de los widgets en sus diferentes plataformas, pasaremos a sintetizar cuáles son las diferencias que se observan entre las mismas.

La primera y más notable diferencia es el soporte que ofrece cada una de las plataformas para distintos sistemas operativos. La siguiente tabla muestra tal información:

	MAC OS X	WINDOWS	OTROS
APPLE	✔	✘	✘
YAHOO!	✔	✔	✘
MICROSOFT	✘	✔	✘
GOOGLE	✔	✔	✔
OPERA	✔	✔	✔

Tabla 1.1: Soporte de widgets en diferentes sistemas operativos

Podemos distinguir por tanto en primer lugar los widgets multiplataforma (ofrecidos por Google y Opera), donde la gestión de widgets se realiza a través de las funciones de un navegador web, permitiendo por tanto su uso en la mayoría de sistemas. Por contrapartida, estos widgets están sujetos a las limitaciones del navegador, siendo imprescindible el mismo para su ejecución, y en la mayoría de casos requieren también de una conexión a Internet.

Por otra parte, aparece una compañía, Yahoo!, que ofrece soporte para los dos sistemas principales, Windows y mac. La ejecución de los widgets se realiza a través de un software específico (Yahoo! widget Engine), portado convenientemente a cada sistema operativo, cuya descarga e instalación son imprescindibles. Una vez realizado tal proceso, resulta necesario descargar los widgets en sí. En este caso, se realiza una distinción entre los widgets, puesto que dependiendo del equipo en que hayan sido desarrollados, serán o no compatibles con la máquina del usuario. Existe también la posibilidad de crear widgets que funcionen en ambas plataformas, aunque su funcionalidad queda limitada.

Por último, nos encontramos con aquellas compañías cuyos widgets son exclusivos para un sistema operativo en particular, y sólo los usuarios del mismo podrán hacer uso de sus ventajas. No resulta necesario descargar software alguno para ejecutar los widgets ya que la aplicación viene integrada en el propio sistema. Los widgets creados en estas plataformas serán exclusivos para las mismas, no es posible compartirlos, a menos que se realicen los cambios pertinentes en su implementación.

Las diferencias a nivel de implementación no son tan notables, puesto que todos los widgets descritos parten de la misma base. De hecho, todos los diferentes tipos de widgets comparten una serie de características básicas, las descritas por las tecnologías web utilizadas: XML y HTML. Por tanto, todos ellos incluyen archivos que definen la estructura de la aplicación mediante un lenguaje de marcas, y las imágenes correspondientes para dar forma a la interfaz. También comparten los lenguajes de programación que dotan de funcionalidad, por norma general JavaScript. Esto propicia la creación de widgets "universales", que pueden funcionar en prácticamente cualquier plataforma modificando muy poco código.

Por consiguiente, los widgets de las diferentes compañías se distinguen entre sí por las características adicionales que proporciona el sistema o aplicación donde se ejecutan. Así, Apple y Microsoft ofrecen las ventajas únicas de sus respectivos sistemas operativos, mientras que Google, Yahoo! y Opera tienen sus propias APIs, que ofrecen diferentes posibilidades al usuario. El resto de diferencias son mas bien sutiles, limitándose a la extensión con que se dota al paquete de cada widget o la manera de definir las propiedades del widget.

La siguiente tabla muestra qué tecnologías emplea cada compañía para crear widgets:

	HTML	XML	CSS	.NET	JS	APPLESCRIPT
APPLE	✔	✘	✔	✘	✔	✔
YAHOO!	✘	✔	✘	✘	✔	✔
MICROSOFT	✔	✘	✔	✘	✔	✘
GOOGLE	✔	✔	✔	✔	✔	✘
OPERA	✔	✘	✔	✘	✔	✘

Tabla 1.2: Tecnologías utilizadas por cada plataforma de widgets

Observamos cómo la gran mayoría se decanta por HTML estándar, a excepción de Yahoo!, que utiliza XML para sus widgets multiplataforma. Además, se aprecia que el lenguaje JavaScript está totalmente consolidado y se usa en todas las plataformas para definir la funcionalidad de los widgets.

Los usuarios

Los widgets son herramientas relativamente fáciles de desarrollar, puesto que se basan de tecnologías de uso muy común. Esto ha conformado la aparición de comunidades de usuarios en las que se comparten las diferentes creaciones de los individuos, así como experiencias y temas de ayuda en lo referente a la implementación.

Apple y Microsoft conforman sus propias comunidades, dedicadas exclusivamente a los usuarios de sus sistemas operativos. Por otra parte, Opera y Google filtran los usuarios según el navegador utilizado, independientemente del sistema operativo que utilicen. Finalmente, Yahoo! ofrece sus servicios a los usuarios de los dos sistemas operativos más comunes.

El siguiente gráfico ilustra los diferentes grupos de usuarios que abarca cada compañía:

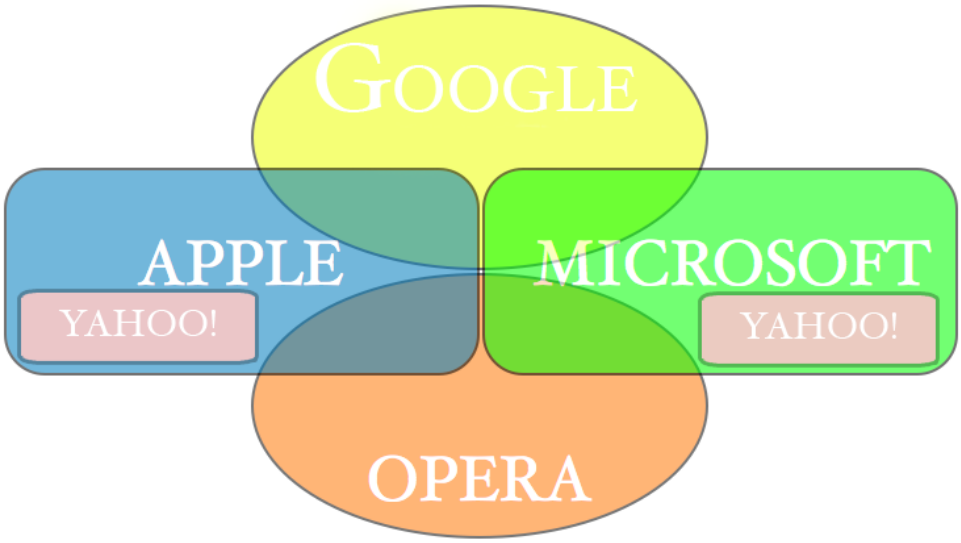


Figura 1.13 : Conjuntos de usuarios comprendidos por las diferentes compañías

Se observa claramente como a partir de dos grandes grupos de usuarios, según el sistema operativo que utilicen, se fragmentan las comunidades de usuarios. Google y Opera atraerán a los usuarios de otros sistemas operativos, pero también a una porción de los usuarios de Apple y Microsoft. Por su parte, Yahoo!, al ofrecer un sistema compatible con Mac OS X y Windows, tomará usuarios de ambas plataformas, creando una única comunidad para todos ellos. Finalmente, Apple y Microsoft cuentan con usuarios que hacen uso exclusivo de sus servicios.

Aquellos que utilizan los últimos sistemas operativos de Apple y Microsoft respectivamente, quizás se decanten por los widgets que ofrecen estas compañías por comodidad, puesto que se les ofrece todo lo necesario para su uso desde el primer momento. Existe también la posibilidad de que se decidan por los widgets de otras compañías, quizás porque sus comunidades resultan más atractivas o amplias, o simplemente porque son más afines con su filosofía. Finalmente, aquellos usuarios de otras plataformas tienen también a su disposición diferentes alternativas a la hora de

utilizar widgets, a través de sus navegadores. No se descarta tampoco el uso simultáneo de widgets procedentes de compañías diferentes, seleccionando lo mejor de cada uno.

¿Se utilizan realmente los widgets? Es una pregunta bastante acertada que nos podemos realizar. Con el fin de contestarla, realicé una encuesta preguntando por los hábitos de los usuarios respecto a los widgets. Hubo un total de 67 participantes. Es evidente que sólo los usuarios regulares de computadores tienen posibilidades de utilizar los widgets, por lo tanto, realicé la encuesta en una comunidad web, ElOtroLado, a la que pertenecen toda clase de Internautas, para intentar obtener unos resultados lo más ajustados posibles a la realidad.

La encuesta consistía en una votación, donde cada usuario tenía que seleccionar la respuesta qué más se adaptara a sus circunstancias. La pregunta era "¿Utilizas Widgets o Gadgets?", y los usuarios podían elegir una de entre 3 opciones: "Los uso muy poco o nada", "Los uso con frecuencia", y "Desconozco su existencia". Además, se animaba a aquellos que quisieran a comentar sobre qué plataforma los utilizaban o habían utilizado.

¿Utilizas Widgets o Gadgets?

	VOTOS	PORCENTAJE
Los uso muy poco o nada	19	28,36%
Los uso con frecuencia	27	40, 30%
Desconozco su existencia	21	31,34%
TOTAL	**67**	**100%**

Tabla 1.3: Resultados encuesta: "¿Utilizas Widgets?"

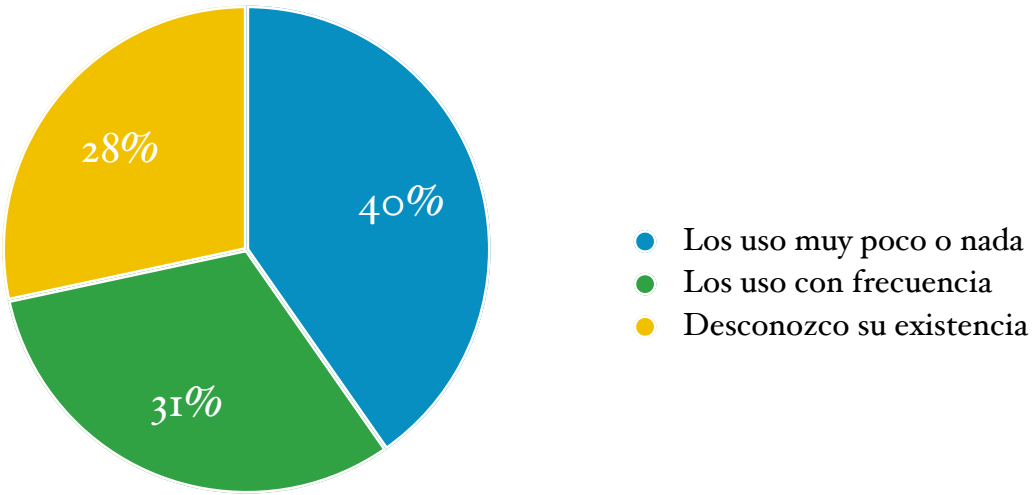


Figura 1.14: Gráfica con sectores del porcentaje de usuarios que utiliza widgets

Aquellos que así lo desearon, tuvieron la oportunidad de comentar bajo que plataforma utilizaban los widgets. La participación en esta parte no fue muy elevada, con lo que los resultados son mas bien poco representativos. A continuación los resultados obtenidos:

Plataforma utilizada para widgets

	VOTOS	PORCENTAJE
Mac OS X Dashboard	7	39%
Windows Vista Sidebar	5	28%
Yahoo! Widgets Engine	3	17%
Linux	3	17%
TOTAL	**18**	**100%**

Tabla 1.4: Resultados encuesta según plataforma utilizada

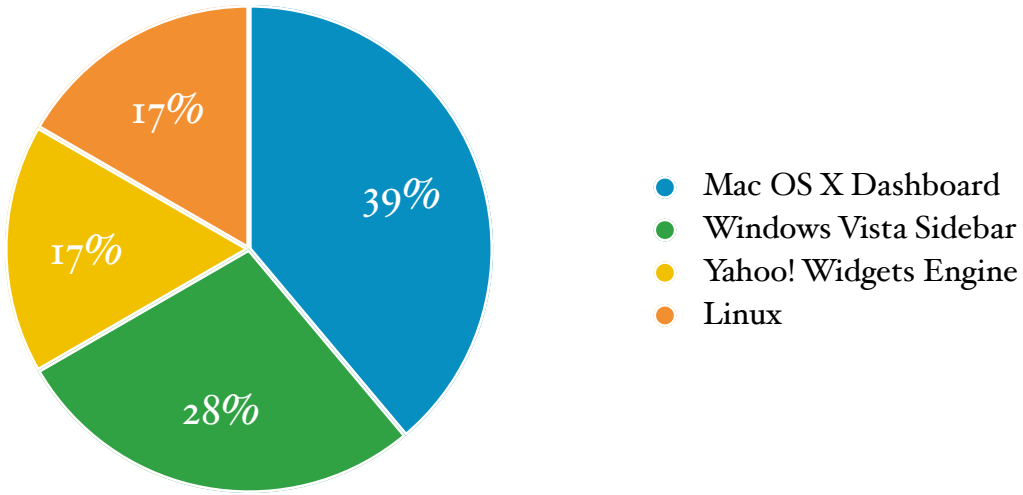


Figura 1.15: Gráfica con sectores del porcentaje de usuarios que utiliza widgets

Antes de pasar a explicar las conclusiones que se pueden extraer de la encuesta, hay que resaltar que los resultados obtenidos se tienen que manipular con mucha precaución, puesto que en primer lugar el volumen de participación es mas bien reducido como para extenderlo a toda la comunidad de usuarios de widgets. Tampoco se puede confirmar de forma alguna la veracidad de los datos y opiniones vertidas, puesto que los usuarios permanecieron en el anonimato. Además, los usuarios sobre los que se realizó la encuesta no son necesariamente representativos de la comunidad en general. Por consiguiente, cualquier conclusión que pudiéramos obtener es meramente informativa y carece de rigor científico.

Dicho esto pasaremos a comentar los resultados. En primer lugar, lo que más llama la atención de la primera gráfica es que el número de usuarios que desconoce el término widget es muy similar al de los que los utilizan o por lo menos los conocen. Este hecho parece indicar que sigue habiendo un gran desconocimiento de estas herramientas, por mucho que se intenten popularizar. Por otra parte, también podría ser significativo el que la respuesta más votada sea "Los uso muy poco o nada". Esto podría apuntar a que, aun a pesar de que los usuarios conozcan los widgets, muchos no los perciben como una herramienta de utilidad y su uso queda limitado a la simple curiosidad, sin ir más allá.

No es posible saber qué plataforma utilizaron estos usuarios en sus fugaces contactos con los widgets, aunque hay mucha probabilidad de que sean aquellos usuarios que utilizan un sistema que no integra widgets. Esto implica que si quieren usar widgets tiene que, en primer lugar conocer de su existencia, a continuación buscar un proveedor de widgets compatible con su sistema, descargarlo, ponerlo en marcha, y por último descargar todos los widgets deseado. No hay lugar a dudas de que muchos no completarían el proceso, puesto que los widgets, a fin de cuentas, no son un elemento imprescindible para los usuarios de computadora. Si el esfuerzo implicado en su obtención es demasiado elevado, muchos simplemente desisten y los ignoran por completo. Sin embargo, aquellos usuarios que dispongan de widgets en su sistema desde el principio, pueden usarlos directamente, sin tener que dar paso alguno, y por tanto son más propicios a darles una oportunidad.

Respecto a la segunda tabla y gráfico, y aplicando las mismas observaciones de cautela, es interesante notar que las plataformas menos socorridas para utilizar los widgets son aquellas que requieren de una descarga para funcionar, lo cual está en armonía con el razonamiento expuesto en el párrafo anterior.

Widgets móviles

Las características propias de los Widgets, uso fácil y rápido y acceso instantáneo para funciones determinadas, los convierten en un elemento ideal para convertirse también en aplicaciones móviles, esto es, integradas en dispositivos tecnológicos portátiles: teléfonos móviles, PDA's, etc. donde el espacio de pantalla está generalmente muy limitado, y la ejecución simultánea de aplicaciones no siempre es fácil o posible. Analizaremos en este apartado cuál es el estado actual de los widgets móviles, sus posibilidades y su capacidad de integración con los widgets de escritorio.

Los widgets están basados en tecnología web, principalmente HTML. Por consiguiente, cualquier plataforma con soporte web puede ser propensa a albergar diferentes tipos de widgets. Este es el caso de los dispositivos móviles. En la actualidad la inmensa mayoría ofrece funciones web, basadas en Web 2.0. Sin embargo, la realidad es que la promesa de Internet móvil no ha cumplido todas las expectativas, y su uso no se ha popularizado. Esto se podría achacar a tres razones, según se señaló en la "Semana del Widget" en Londres (Widget Week 2007):

- Coste elevado - De momento, sigue siendo caro navegar por Internet a través de un dispositivo móvil (teléfonos mayormente), a pesar de los esfuerzos de las compañías por ofrecer ofertas interesantes.
- Interfaz no atractiva - Generalmente la interfaz de usuario tanto de los navegadores como de las aplicaciones móviles deja mucho que desear, tanto a nivel práctico como visual.
- Pobre experiencia de usuario - La navegación sigue siendo a día de hoy una tarea difícil y aparatosa, no se ha logrado suplir de forma satisfactoria la ausencia de un teclado completo y el reducido tamaño de la pantalla dificulta también la comprensión de los mensajes y opciones de navegación.

Es en este punto donde los widgets móviles entran en juego. La posibilidad de integrar las necesidades de servicios web más comunes en estas pequeñas aplicaciones, de diseño simple pero efectivo, vienen a suplir las deficiencias que adolecen los navegadores de dispositivos móviles. Este tipo de widgets resulta de interés tanto para el usuario final, que puede elegir y personalizar su propia colección de widgets, como para el desarrollador, capaz de crear y utilizar fácilmente aplicaciones para usuarios finales en forma de widgets. Es evidente que no todas las aplicaciones tienen sentido en el contexto de los widgets, pero, sí se pueden cubrir gran parte de las necesidades.

Tal y como se menciona en la primera parte del documento, para poder ejecutar un widget, es necesario un "motor Widget", que podemos entender como el marco de trabajo necesario para hacer funcionar estas micro aplicaciones. En el campo móvil, son varias las compañías que han desarrollado sus propios "motores", entre ellas: Widsets, Bluepulse, Opera, etc. Estos motores están disponibles sobre la plataforma J2ME (Java 2

Platform Micro Edition), lo cual permite que los desarrolladores usen Java para crear diferentes aplicaciones sobre dispositivos móviles. Cada una de estas compañías ofrece su propio SDK para desarrollar widgets.

A día de hoy, los widgets móviles son una realidad, pero eso sí, limitada de momento. No todos los widgets convencionales están disponibles para dispositivos móviles. De hecho, sólo hay un subconjunto de widgets que actualmente funcionan, que son los siguientes:

- RSS
- Flickr
- Mail
- Notas
- Meteorología
- Reloj
- Fondos

Figura 1.16: Ejemplos de widgets móviles

El problema es que de momento, no es posible usar un navegador web para ejecutar los widgets, en primer lugar porque son muy pocos los terminales con la capacidad para ejecutar widgets complejos dentro del marco de un navegador y en segundo lugar porque su utilización en dispositivos móviles no se ajusta a las características de los navegadores.

Hasta el momento, diferentes compañías desarrollan sus propios motores y widgets específicos para los mismos, lo que conduce a widgets que sólo funcionan en determinados terminales o con determinados servicios. Ha sido Opera la que ha dado el primer paso hacia una posible estandarización de los widgets, enviando las especificaciones de un widget a la W3C para asegurarse de que los widgets móviles funcionan en cualquier plataforma. Este paso abre la posibilidad de trabajo cooperativo con el resto de compañías para establecer un estándar de widget que todos implementen. De esta manera, los desarrolladores podrán dedicarse sencillamente a crear widgets, siendo el usuario quien decida la plataforma sobre la que utilizarlos.

¿Qué se puede esperar en el futuro? Es evidente que aún se puede avanzar mucho en este campo. Se esta trabajando mucho en esta área, y la investigación avanza de forma paralela a las tecnologías vinculadas a los widgets. Uno de los objetivos primordiales es acercar los widgets al usuario final, dándolos a conocer, y facilitando en la medida de lo posible su utilización, tratando de alcanzar compatibilidad total con los terminales del mercado. Todo apunta a que la interoperabilidad será la clave, poder llevar tus widgets a cualquier sitio, sin importar el dispositivo o plataforma. También cobrará importancia la comunicación entre widgets, con todas las ventajas que ello supondría.

Widgets y el software libre: Desklets

Hasta el momento hemos cubierto las principales compañías comerciales que proporcionan widgets entre sus servicios. Los usuarios deberán decantarse por una u otra para poder usarlos. ¿Qué pasa con los usuarios de software libre? Según lo visto hasta el momento, la única solución para esta comunidad en lo que ha widgets se refiere sería utilizarlos a través de las respectivas versiones de los navegadores de aquellas compañías que ofrecen los widgets de esta forma (widgets universales). Esta solución, sin embargo, a buen seguro no será del agrado de todo los usuarios, puesto que se verían forzados a usar un navegador comercial y a tener una conexión a internet operativa para poder utilizar estas pequeñas aplicaciones. ¿Existe alternativas en el mundo del software libre? Por supuesto.

Los widgets se conocen en el entorno de la programación libre como "Desklets". Existen varios programas o paquetes que sirven estos widgets en sistemas operativos de libre distribución, como por ejemplo:

- gDesklets (http://gdesklets.org/)
- SuperKaramba (http://netdragon.sourceforge.net/).
- GKrellM (http://www.gkrellm.net/)
- aDesklets (http://adesklets.sourceforge.net/

Existe varias diferencias entre los programas, básicamente en términos de rendimiento, dependencias de librerías o entornos de escritorio requeridos, aunque todos cumplen con su función.

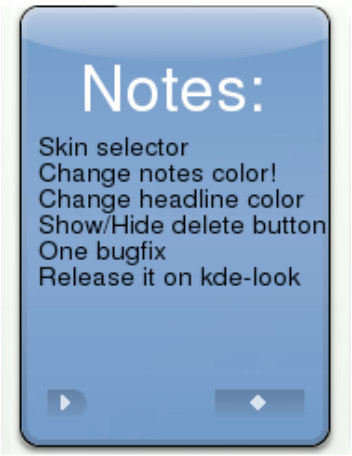


Figura 1.17: Desklet bloc de notas, (SuperKaramba)

Figura 1.18: Desklet calendario (gDesklets)

Figura 1.19: Desklet carga CPU (aDesklets)

La base de los Desklets es muy similar a la de los widgets. Siendo el resultado casi idéntico (mostrar pequeñas aplicaciones en el escritorio), el proceso de desarrollo e implementación resulta ligeramente diferente, aunque se basa en los mismo principios. Para ejecutar un desklet, es necesario en primer lugar disponer de un proceso servidor de Desklets, un Daemon que se encarga de gestionar el uso de los mismos en el escritorio. Así pues, tendremos que descargar e instalar un servidor de desklets de entre los disponibles. El siguiente paso será descargar los desklets en sí e incorporarlos a nuestro sistema con el fin de ejecutarlos.

En lo que a implementación se refiere, las diferencias son más palpables. Para empezar, los desklets no están construidos a partir de HTML, como los widgets, sino que se diseñan a partir de las librerías del sistema. El lenguaje de programación por excelencia utilizada para los desklets es Python, seguido muy de cerca por Perl. En lo que a formato y estilo se refiere, tampoco se hace uso de CSS, sino que se suelen definir en un archivo de extensión XML, posteriormente interpretado por un determinado lenguaje de programación basado en XML (e.g. ADL: Applet Description Language).

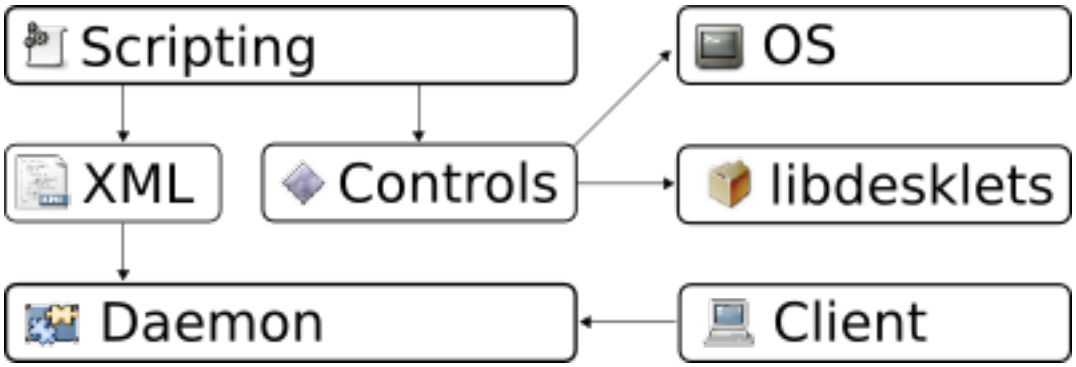


Figura 1.20: El entorno de trabajo de una de las distribuciones de desklets: gDesklets

En este caso, la estructura utilizada para usar widgets es la siguiente:

- Un daemon se ejecuta en segundo plano, actuando como servidor para proporcionar un canal de comunicación Los clientes (el usuario) se conectan al mismo mediante un socket para darle instrucciones (e.g. abrir un desklet)
- El daemon es responsable de cargar y ejecutar los desklets. Los clientes (e.g. linea de comandos) se pueden entonces implementar como controles remotos estilizados.
- La interfaz de usuario de los desklets es definida por ADL, lenguaje basado en XML
- Se puede dotar de interactividad a la interfaz de usuario mediante scripts, como por ejemplo Python, incrustados directamente en el código XML o importados.

En resumen, los usuario de software libre tienen a su disposición posibilidades variadas para utilizar "Widgets" en sus terminales sin tener que recurrir a soluciones comerciales. Sin embargo, lo cierto es que el número de desklets desarrollados y actualmente disponibles en la red no es muy abundante, sobre todo al compararlo con el catálogo de que disponen sus más inmediatos competidores. Las causas podrían ser atribuidas a la elevada complejidad asociada al desarrollo de los desklets (requieren de tecnologías y procesos menos convencionales y conocidas) o a la poca popularidad de estas mini-aplicaciones entre los usuarios. Además, por norma general la estética de los mismos no suele estar tan cuidada (también en parte debido a la mayor dificultad para dar estilo y formato a las imágenes y contenidos).

En cualquier caso, son una alternativa útil a los widgets y deberían ser tenidos en cuenta por los usuarios, puesto que gozan de todas las ventajas que ofrece el software libre: libre distribución de contenidos y comunidades de usuarios que se ayudan mútuamente.

Tecnologías Envueltas

Dashcode

Dashcode es una aplicación de reciente aparición cuyo principal objetivo es facilitar en la medida el desarrollo e implementación de widgets. Esta herramienta formará parte del nuevo sistema operativo de Apple: Leopard. Sin embargo, ha sigo generada con compatibilidad hacia atrás, esto es, se podrá utilizar en versiones anteriores del sistema operativo, concretamente en Tiger. Todavía no está disponible la versión final de Dashcode, puesto que hará su aparición definitivo de forma conjunta con Leopard, aunque de un tiempo a esta parte, Apple ha facilitado a los desarrolladores una beta del programa, compatible con Mac OS X Tiger. Esta beta es la versión que he utilizado para desarrollar el widget que nos ocupa.

Antes de existir Dashcode, el proceso de creación de un widget se realizaba siguiendo los pasos descritos en el apartado "Diseño e Implementación en las diferentes plataformas" (y de hecho esta sigue siendo la opción preferida por muchos desarrolladores, especialmente por los que utilizan widgets de otras plataformas). Este proceso, requiere del uso de uno o varios editores de texto para generar los archivos de código, programas de edición gráfica para las imágenes y botones, navegadores web para comprobar los resultados e incluso interpretes para garantizar el correcto funcionamiento del código utilizado.

Una vez obtenidos todos los archivos e imágenes necesarias, era necesario crear el paquete de extensión .wdgt (o la necesaria según el sistema) para poder incorporar el widget en el dashboard. Sólo entonces se podía verificar el comportamiento del widget. En caso de que no fuera el correcto, había que volver a modificar los archivos uno por uno, eliminar el widget del dashboard y volver a crear el paquete, y así sucesivamente hasta obtener una versión definitiva.

Dashcode aparece con el objetivo de simplificar enormemente todos estos pasos y operaciones envueltos en la creación de widgets. Partiendo de la base de que es posible automatizar gran parte del código que se necesita, Dashcode ofrece un entorno de trabajo común para todas las diferentes fases de creación de un widget, poniendo a disposición del usuario una interfaz gráfica para crear los widgets, generando el código de forma automática según las acciones del usuario. Permite editar código, modificar las capas del widget, realizar pruebas en cualquier momento, depurar el código, etc. Además, gestiona automáticamente todos los archivos necesarios, creándolos o eliminándolos según sea necesario.

También resulta muy útil a la hora de probar el widget, ya que ofrece la posibilidad de ejecutarlo en cualquier momento, con tan sólo apretar el botón "play", simulando su comportamiento en el Dashboard. Esto agiliza en gran manera la creación de los widgets, y da lugar a un método de trabajo mucho más flexible y rápido.

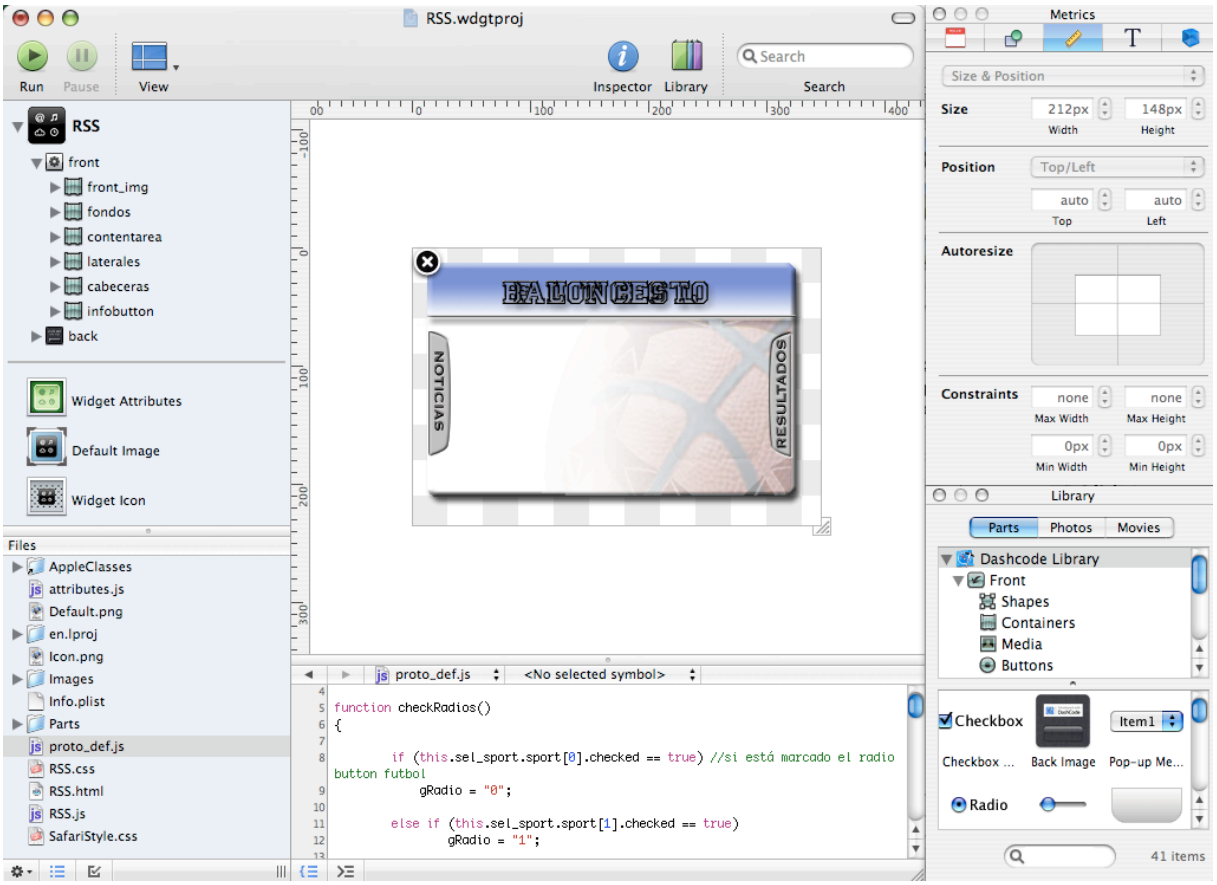


Figura 1.21: Captura de Dashcode en pleno funcionamiento. En el lado izquierdo está el navegador, que permite seleccionar la herramienta deseada durante el desarrollo del widget. La parte principal de la ventana contiene el lienzo (canvas), donde se compone la interfaz del widget. La ventana superior derecha muestra el "Inspector", que permite modificar los atributos del objeto seleccionado, y justo debajo la ventana "Library", con diferentes elementos que se pueden añadir al widget. En la parte inferior, un editor de texto para visualizar o modificar los archivos de código

Otra de las grandes ventajas que ofrece Dashcode, aparte de la automatización de código y la gestión de archivos, es la posibilidad de crear un proyecto a partir de una plantilla. La aplicación ofrece por defecto varias plantillas, cada una se corresponde con un tipo de widget, y permite crear un widget desde cero en muy poco tiempo y sin escribir una sola línea de código. Otra posibilidad es utilizar una plantilla para iniciar el proyecto, y ampliar su funcionalidad retocando el código pre-escrito por Dashcode o añadiendo código propio.

¿Significa esto que el desarrollador puede olvidarse por completo de la forma de funcionamiento de un widget? Depende. Un usuario que decida crear un widget sin demasiadas pretensiones, no necesitará conocer ni entender el código que Dashcode ha generado. Sin embargo, todo desarrollador avanzado que desee controlar todos los aspectos de su widget tendrá la necesidad de conocer las especificaciones de las diferentes clases de Apple, así como del lenguaje HTML y JavaScript, con el fin de ajustar el código auto-generado según sus preferencias o necesidades.

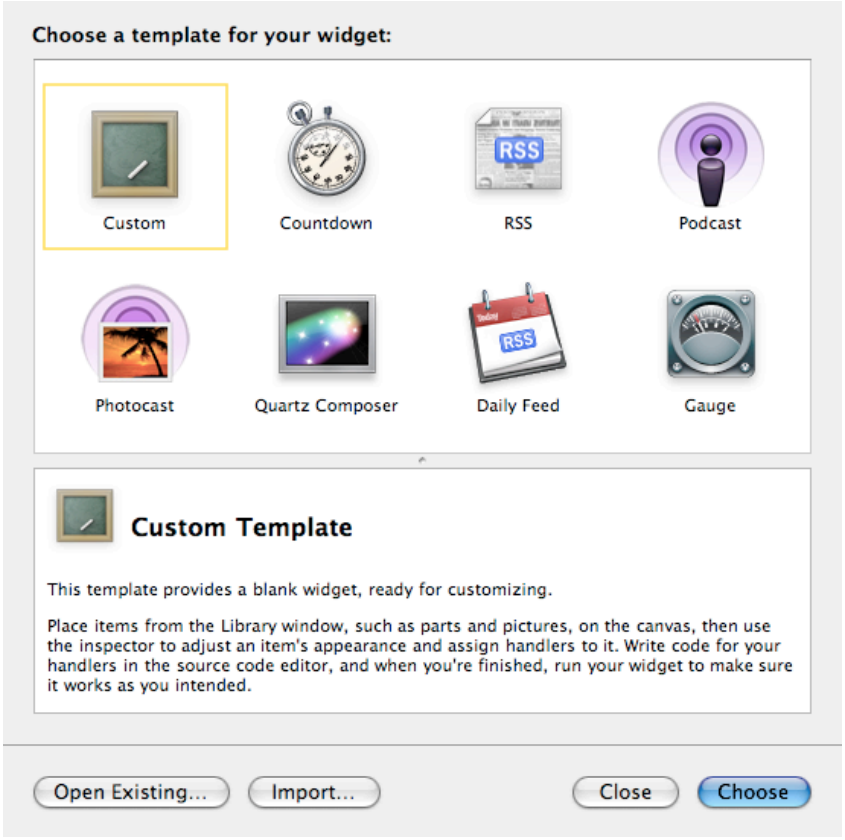


Figura 1.22: Pantalla de selección de plantillas para iniciar un proyecto con Dashcode

En definitiva, Dashcode resulta una aplicación extremadamente útil a la hora de desarrollar widgets, gracias a su versatilidad y facilidad de manejo. Esto se traduce en un aumento en la calidad de los widgets, puesto que permite a los desarrolladores centrar sus esfuerzos en otros aspectos como la eficiencia, funcionalidad nueva, etc., sin tener que preocuparse por generar código que "ya ha sido inventado".

Desafortunadamente, de momento no existe una alternativa similar para widgets de otras plataformas, con lo que sólo los desarrolladores de widgets para Mac podrán beneficiarse por el momento de sus virtudes. Existe también la posibilidad de generar el widget con Dashcode y portarlo posteriormente a las plataformas deseadas, puesto que generalmente son pocos los cambios necesarios.

Es cierto por otra parte que la versión beta de Dashcode con la que elaboré el widget de este trabajo no funcionaba todo lo bien que cabría esperar de una aplicación de estas características. Los cuelgues, bloqueos, etc. fueron bastante numerosos, así como otros fallos, aunque es de esperar que sean debidamente subsanados en la versión definitiva. En cualquier caso, Dashcode es una gran ayuda para desarrollar widgets, a buen seguro tendrá una gran aceptación en el futuro.

JAVASCRIPT

JavaScript, es un lenguaje de programación interpretado, incorporado en la mayoría de navegadores web, creado originalmente por Netscape Communications y Sun Microsystems. Fue diseñado para dotar de interactividad a los documentos HTML y páginas web en general. No sólo eso, sino que también permite generar páginas web de forma dinámica. Tal y como ya se ha mencionado, un widget no es más que una página web de proporciones muy reducidas, con lo que JavaScript es el lenguaje ideal para definir el comportamiento de nuestros widgets.

El hecho de que la mayoría de widgets no necesitan de una conexión a Internet para funcionar (a pesar de que puedan ser entendidos como páginas web), convierte a JavaScript en el lenguaje ideal para acoplar al código html, puesto que es ejecutado en el navegador del usuario (en este caso en el servidor de widgets, o en los propios navegadores en el caso de los widgets universales).

JavaScript, es orientado a prototipos, y permite trabajar con la jerarquía de objetos de una página web, tal y como establece el DOM (Document Objet Model). A lo largo de la creación del widget haremos constante referencia a esta jerarquía. A continuación se muestra su estructura:

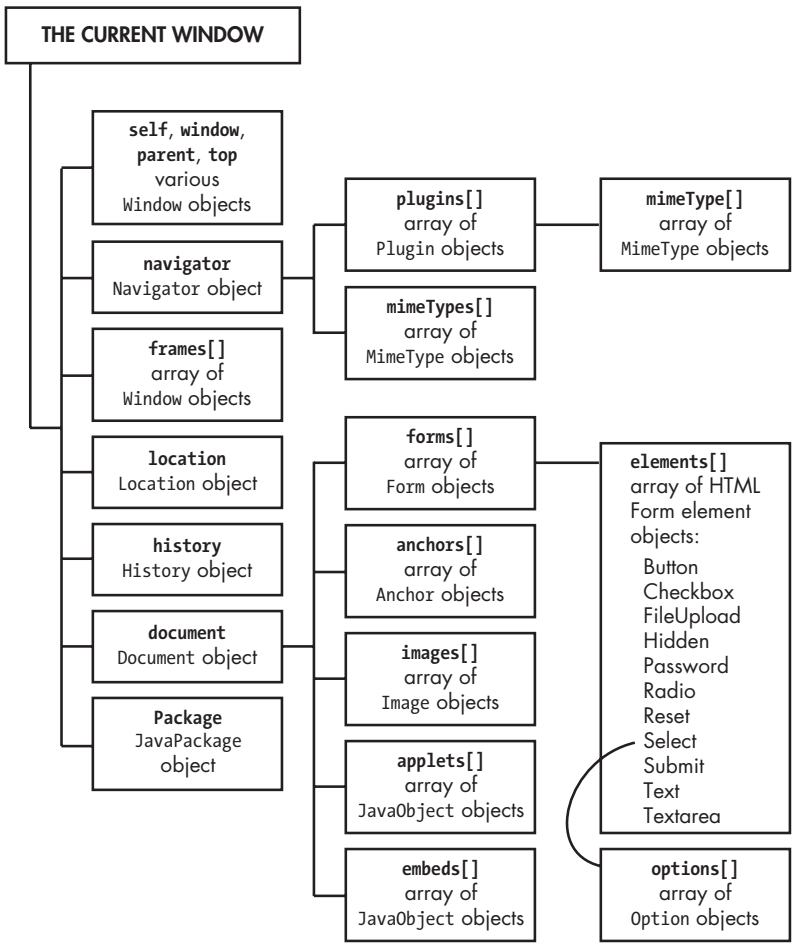


Figura 1.23: Organización jerárquica del DOM

JavaScript permite acceder y modificar muchas de las propiedades de los objetos de la jerarquía, e incorpora también una serie de métodos de gran utilidad. Los métodos y propiedades utilizadas para la implementación del widget en este proyecto se comentarán en la sección pertinente.

El uso de JavaScript para la creación de widgets tiene ciertas peculiaridades. Para empezar, lo que en general se consideran desventajas de JavaScript, como por ejemplo el hecho de que su comportamiento sea diferente según el navegador utilizado, no tiene efecto alguno en los widgets, ya que serán utilizados siempre con el mismo servidor de widgets. Por otra parte, el que no permita crear gráficos tampoco resulta un gran inconveniente, puesto que sí tiene gran versatilidad para manipular imágenes ya existentes, siendo este nuestro caso. Finalmente, para interactuar con servidores de internet utilizaremos el objeto `XMLHttpRequest`, inherente en los navegadores, puesto que JavaScript, como la mayoría de lenguajes de programación, no incorpora de forma nativa las herramientas necesarias para ello.

CANALES RSS

RSS es un estándar (Really Simple Syndication, RSS 2.0), un formato de datos basado en XML que facilita la redifusión de contenidos de un sitio a web a sus suscriptores (pueden ser tanto otros portales de internet como usuarios particulares). Cuando se produce una actualización en forma de archivo RSS en una web cualquiera, todos aquellos suscritos al canal RSS reciben la nueva información de inmediato en forma de titulares o resúmenes, sin necesidad de navegar por la página web hasta encontrar la información deseada o comprobar regularmente si han aparecido nuevas noticias o artículos.

La obtención de los nuevos datos se realiza mediante agregadores o lectores de canales RSS, que automáticamente comprueban los portales web indicados y obtienen la nueva información publicada, mostrándola al usuario de forma organizada. Los agregadores suelen estar integrados en los propios navegadores de internet, aunque también se pueden utilizar por separado.

Esta tecnología resulta particularmente útil en el mundo de los widgets, ya que se adapta perfectamente a su filosofía: proporcionar al usuario una determinada función o información de forma rápida y transparente. De hecho, haremos uso de esta tecnología en el apartado de implementación para obtener de la web las últimas noticias y resultados del mundo deportivo y mostrarlas en el widget. El hecho de que sea una tecnología basada en XML también supone una ventaja, puesto que permite procesar la información obtenida y adaptar tanto su estilo como su extensión.

Figura 1.24: Icono que denota la presencia de un canal RSS. Se ha convertido en un estándar de facto

II - BLOQUE DE DESARROLLO

Etapas del proceso de implementación

Elección de un Widget

En primer lugar se tuvo que decidir la temática y funcionalidad del Widget a implementar. El único requisito a priori era que hiciera uso de servicios web, y que cumpliera con las reglas básicas de todo widget: utilidad para el usuario y a la vez simplicidad en su diseño.

Tal y como se ha explicado en el bloque de estado del arte, en la actualidad existen un gran número de Widgets diferentes. Este hecho dificulta en gran manera la tarea de encontrar un widget novedoso y útil que no haya sido implementado todavía.

Para afrontar esta primera barrera, recurrí a la técnica "brainstorming", anotar cualquier idea o tema que pudiera traducirse en un widget, sin importar la viabilidad u originalidad del mismo. Una vez conseguida una lista con diferentes ideas, y tras descartar todos los que parecieran inviables o ya inventados, realicé una serie de bocetos en papel de los candidatos, tratando de plasmar las opciones que se ofrecerían en ambas caras de cada widget y las transiciones entre las mismas.

Contra todo pronóstico, surgieron un buen número de posibles widgets que podrían ser objeto de trabajo. En la página siguiente se muestran en forma de boceto algunas de las ideas, todas ellas candidatas a ser transformadas en un widget. La decisión final no fue fácil. Consulté las opiniones de diferentes personas respecto a las diferentes ideas obtenidas, y finalmente me decanté por un widget de deportes. Su aspecto, funcionalidad y objetivos se detallan más adelante. A grandes rasgos, el widget permitirá al usuario consultar las últimas noticias relacionadas con su deporte favorito (los deportes concretos que se incluirían no se habían decidido completamente en esta etapa, aunque fútbol y baloncesto aparecían entre los favoritos), así como los resultados de los partidos más recientes.

En realidad ya existen varios widgets con esta misma temática, pero están muy localizados (sólo permiten consultar los datos sobre las competiciones y deportes de determinados países), con lo que decidí elaborar una versión que cubriera las necesidades en este aspecto de la península ibérica, donde la afición al deporte está fuera de toda duda.

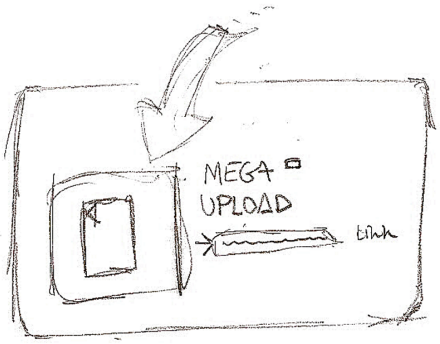


Figura 2.1: Boceto de un widget que permitiría subir archivos al servidor www.megaupload.com directamente, sin entrar en la página, retornando un enlace al archivo subido

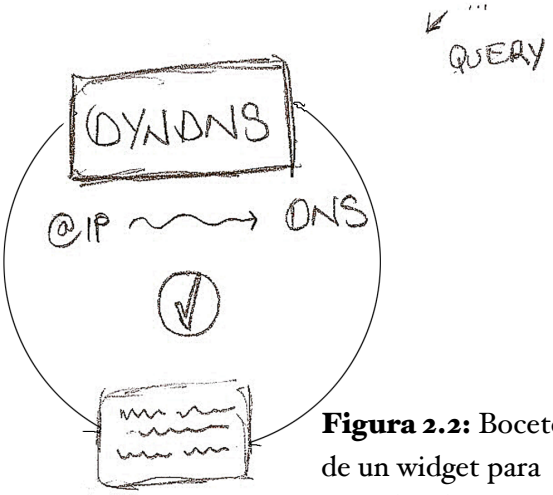


Figura 2.2: Boceto de un widget para consultar los dominios asociados a direcciones IP dinámicas

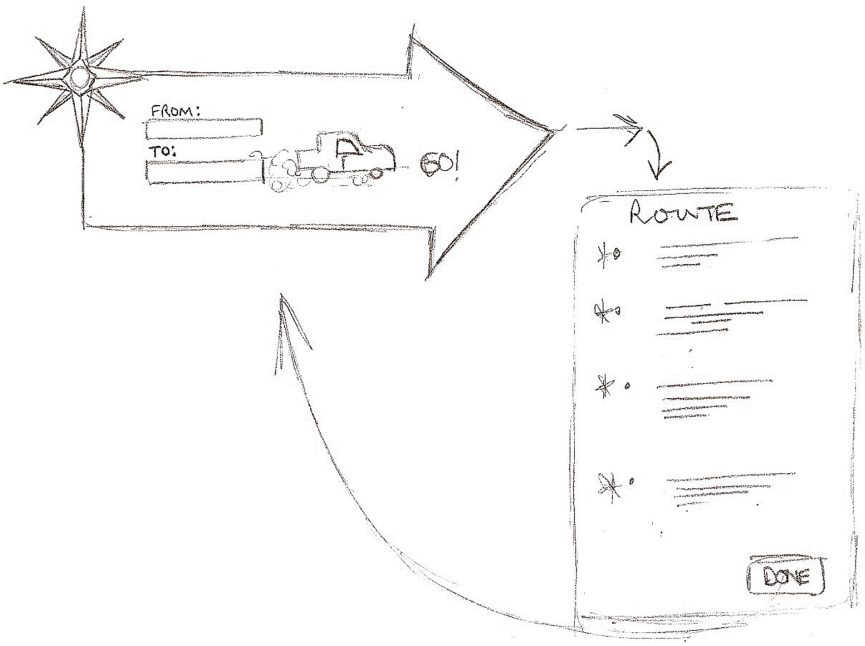


Figura 2.3: Boceto de un widget para mostrar al usuario la ruta de carretera a seguir para alcanzar el destino introducido, desde el origen deseado.

Figura 2.4: Boceto de un widget para localizar los paquetes de correos a partir de su código de paquete.

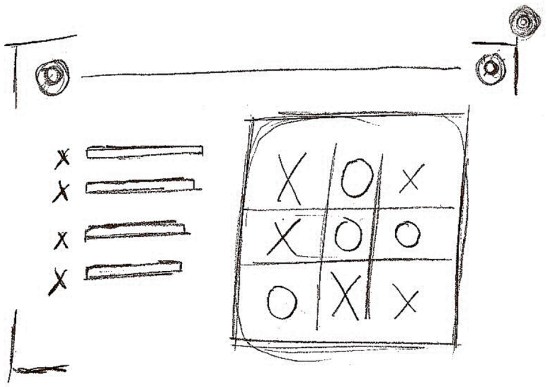


Figura 2.5: Boceto del juego de 3 en raya en forma de widget, con la salvedad de que pretendía ofrecer juego online, partidas contra otros jugadores a través de internet

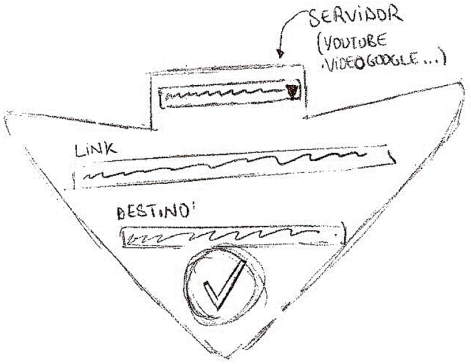


Figura 2.6: Boceto de un widget para descargar vídeos de diferentes servidores (youtube.com, video.google.com, etc;) en la máquina del usuario

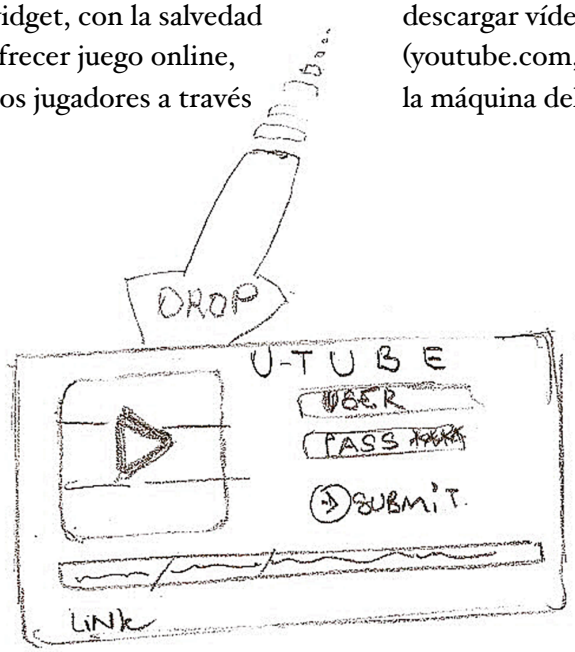


Figura 2.7: Boceto de un widget que pretendía facilitar la tarea de subir vídeos a www.youtube.com permitiendo arrastrar nuestros vídeos sobre el widget y obtener directamente un link al mismo una vez subido

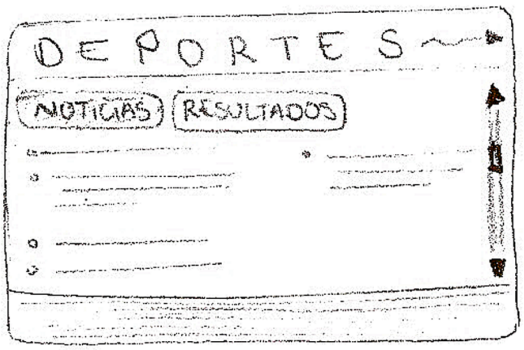


Figura 2.8: Boceto de un widget para consultar rápidamente las últimas noticias y resultados del mundo del deporte, acabó siendo el candidato ganador.

PROTOTIPADO

Esta segunda fase consistió en elaborar una serie de prototipos iniciales de la interfaz gráfica del widget, tanto del anverso como del reverso. Realicé los diferentes prototipos con la ayuda de varias herramientas de edición gráfica (The Gimpshop, Adobe Photoshop CS, entre otros), aunque los bocetos de los prototipos iniciales los hice sobre el papel Explicaremos el proceso de creación del prototipo definitivo, puesto que todas las versiones elaboradas se realizaron de una forma muy similar.

En primer lugar había que crear un fondo para el widget, una forma básica sobre la que mostrar los contenidos. Atendiendo a la guía de estilos de Apple, y teniendo en cuenta las necesidades del widget que nos ocupa, hice un forma rectangular con los bordes redondeados, con unas dimensiones muy ajustadas (330 x 220 px), de forma que no ocupe demasiado espacio. Sobre la capa apliqué diferentes propiedades de fusión de capa: *Drop Shadow*, (modo: normal, 100% de opacidad), *Bevel and Emboss* (estilo Inner Bevel, técnica Smooth, profundidad 100%, Dirección Up, tamaño 5 px.). Apliqué un degradado parcial a la parte superior, donde se mostrará un título (deporte seleccionado), y finalmente inserté un separador (una linea horizontal con las siguientes propiedades de fusión de capa: *Drop Shadow* (modo normal, 100% de opacidad), *Bevel and Emboss* (estilo Inner Bevel, técnica Smooth). El separador diferencia la zona de contenidos con la zona de título de sección. El resultado hasta el momento se aprecia en la figura 2.9.

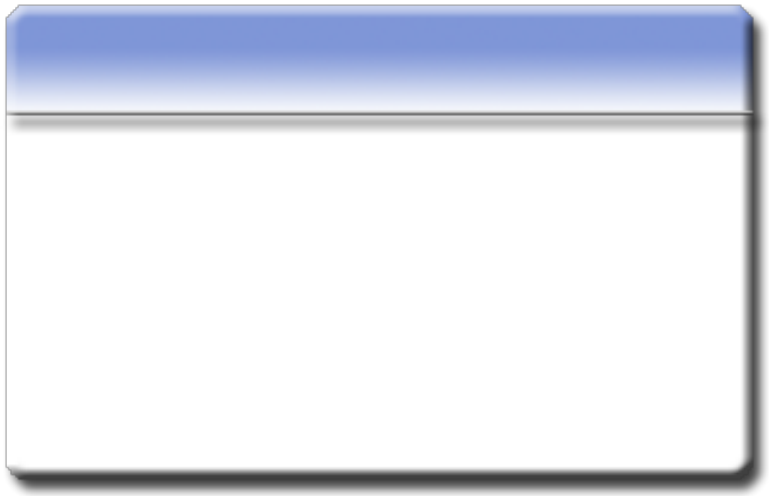


Figura 2.9: Anverso del widget en la primera fase de prototipado

Haciendo uso de las posibilidades que ofrecen las aplicaciones de edición gráfica, trabajé todo el prototipo sobre un mismo archivo, añadiendo los diferentes contenidos en nuevas capas. La ventaja de trabajar con capas es que posteriormente puedes aislarlas y guardarlas en formato de imagen por separado, ideal para construir el widget. Con la capa de fondo ya diseñada, el siguiente paso fue añadir diferentes motivos decorativos, de acuerdo con cada deporte. Obtuve varias imágenes relacionadas con los deportes, y les apliqué las siguientes transformaciones:

-Recortar imagen (ajustarla al tamaño de la zona de contenidos del fondo)

-Propiedades de fusión de capa: *Gradient Overlay* (*Blend Mode*: Normal, *Opacity*: 96%, *Scale*: 66%), *Angle*: 0°, *Style*: Linear).

Obtuve dos capas diferentes (una por deporte, existiendo la posibilidad de añadir nuevas capas en el futuro ante posibles expansiones de contenidos). El resultado de fusionar las capas obtenidas hasta el momento genera los siguientes resultados:

Figura 2.10: Prototipo de los anversos del widget en cada modo diferente

A continuación, y siempre en capas nuevas, procedí a crear los botones de interacción, que serán dos: "Noticias", para consultar las noticias más recientes relacionadas con el deporte seleccionado, y "Resultados", para mostrar los resultados de los últimos partidos disputados. Tras hacer varias pruebas, decidí colocar los botones a ambos lados del widget, en forma de pestaña vertical. Para crear los botones, seguí el siguiente procedimiento:

- Crear una ruta de selección para definir la forma de los botones. Esto lo hice con la herramienta "Pluma".
- Rellenar el contenido de la selección del color deseado (en este caso, gris: "C2C1C1")
- Aplicar las siguientes propiedades de fusión de capa: *Bevel and Emboss* (*Style*: Pillow Emboss, *Technique*: Smooth, *Depth*: 100%, *Direction*: Up, *Size*: 1px, *Soften:* 0px.)
- Duplicar la capa del botón e invertirla horizontalmente para crear el botón opuesto.
- Crear dos nuevas capas, de texto en esta ocasión, para mostrar las etiquetas de cada botón. La fuente utilizada fue: Copperplate Regular, 14pts, suavizada. A cada capa de texto se aplicaron las siguientes propiedades de fusión de capa: *Bevel and Emboss* (*Style*: Pillow Emboss, *Technique*: Smooth, *Depth*: 1%, *Size*: 0px, *Soften*: 0px).
- Duplicar las dos últimas capas, modificando el color de los botones, para diferenciar cuál de ellos está seleccionado. El nuevo color para el botón seleccionado es "BDDCC7"
- Finalmente, sólo restaba desplazar cada capa para ajustarla con el fondo en la posición deseada, situando las capas correspondientes a los botones seleccionados bajo los botones sin seleccionar, quedando ocultos a la vista.

El aspecto de los botones tras el proceso detallado se muestra a continuación:

Figura 2.11: Prototipo de los botones de selección de la parte frontal

El resultado de insertar los botones en el fondo creado anteriormente es el siguiente:

Figura 2.12: Muestra de la parte frontal con botones en fase de prototipado

Antes de proseguir con el acabado, es muy importante también disponer de un botón que permita acceder a la parte posterior del widget. En este aspecto decidí seguir los estándares propuestos por Apple, así que opte por incluir el botón de información que usa Apple por defecto, haciendo uso de las clases que se encuentran a disposición del usuario. Se detalla el proceso en el apartado de implementación, puesto que la inclusión de este elemento no requiere de una fase de prototipado.

El último paso para finalizar el prototipo del widget, fue crear los diferentes títulos que aparecerán en la parte frontal. Los pasos a seguir:

- Crear una capa nueva para cada título. Insertar texto correspondiente a cada capa ("Futbol" y "Baloncesto" respectivamente). Fuente: PrinceTown LET, tamaño: 30 px.
- Aplicar sobre la capa varias propiedades de fusión de capa: *Drop Shadow* (*Blend Mode*: Multiply, *Opacity*: 35%, *Angle*: 120°, *Distance*: 5px, *Spread*: 0px, *Size*: 5px), *Bevel and Emboss* (*Style*: Inner Bevel, *Depth*: 1%, *Size*: 0px, *Soften*: 0px), *Contour* (*Range*: 50%)

La apariencia de los títulos tras las transformaciones:

Figura 2.13: Cabeceras diseñadas para insertar en la parte frontal del widget

La parte frontal del prototipo ya está lista, y ha quedado así:

Figura 2.14: Versiones finales de la interfaz frontal del widget para cada deporte

A continuación se detalla el proceso seguido en el diseño y creación de la parte trasera, el reverso del widget.

En este caso el procedimiento es mucho más sencillo, puesto que la parte trasera tiene un diseño más simple, sencillamente permitirá seleccionar un deporte. Un requisito importante es que tenga una forma y tamaño idénticas a la parte frontal, para dar sensación de continuidad al girarlo. Por consiguiente, el primer paso fue duplicar la capa del fondo de la parte frontal. Pinté esta nueva capa de color blanco ("FFFFFF") y apliqué las siguientes propiedades de fusión de capa: Bevel and Emboss (Style: Inner Bevel, Depth: 51%, Size: 0px, Soften: 8px, Shadow Mode: Multiply, Opacity: 64%), Satin (Blend Mode: Darken, Opacity: 11%, Angle: 19°, Distance: 18px, Size: 35px).

El resultado:

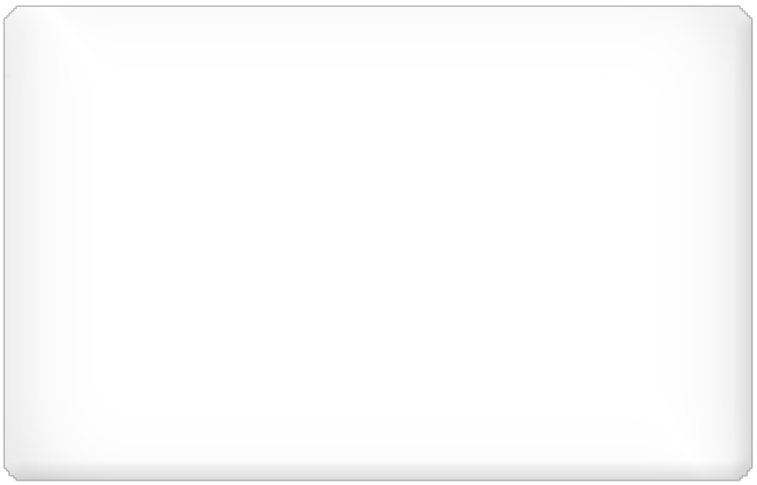


Figura 2.15: Imagen de fondo utilizada para la parte posterior

Sobre el fondo, sencillamente dispondremos un formulario HTML que permitirá seleccionar una de las diferentes opciones, un mensaje para el usuario y un botón para confirmar y volver a la parte frontal.

La creación del botón se realizó manteniendo el estilo seguido hasta el momento. El procedimiento que seguí fue el siguiente:

- Mediante la herramienta pluma hice un trazado del contorno del botón, dándole la forma deseada (en este caso rectangular con bordes redondeados).
- Convertí la ruta en una selección y puse a modo de fondo un degradado reflejado al 100% de opacidad, combinando los colores blanco y gris ("FFFFFF" y "D4D4D4"). Apliqué a la capa estas propiedades de fusión de capa: Bevel and Emboss (Style: Inner Bevel, Technique: Smooth, Depth: 41%, Size: 3px, Soften: 5px)
- En una nueva capa de texto, escribí el contenido del botón, "Volver", usando la fuente Helvetica Neue Regular, con un tamaño de 18 pixeles y de color rojo oscuro ("574F4F"). A esta misma capa apliqué varias propiedades de fusión de capa para dar sensación de relieve al texto: Bevel and Emboss (Style: Emboss, Technique: Smooth, Depth: 31%, Size: 6px).
- También había que contemplar el aspecto que tendría el botón al ser apretado, así que dupliqué la anterior capa de texto, manteniendo el estilo de la capa, pero añadiendo las siguientes propiedades: Inner Glow (Blend Mode: Screen, Opacity: 80%, Noise: 0%, Color: "F5D210").

- El resultado son dos botones, el botón por defecto, y el botón seleccionado, como resultado de fusionar la capa del fondo del botón con cada una de las capas de texto. Se muestra el resultado en la siguiente figura.

Figura 2.16: Prototipo del botón "Volver" en sus dos fases de selección

Añadí además un pequeño mensaje de texto para solicitar al usuario la selección de un deporte ("Selecciona un Deporte"). El proceso de creación del mensaje será muy similar al de pasos anterior: Nueva capa de texto (Fuente Helvetica Neue, 18 px, color negro), propiedades de fusión de capa (Bevel and Emboss (Style:Emboss, Depth:21%).

La parte posterior del widget, queda finalmente así:

Figura 2.17: Versión preliminar de la parte trasera del widget

El formulario que permitirá seleccionar los deportes se incluirá en el momento de la implementación, puesto que se crea a partir de código HTML.

Con una versión del prototipo más o menos definitiva finalizada, ya se podía empezar a implementar el widget, dotándolo de funcionalidad. El proceso se detalla en la correspondiente sección de Implementación.

IMPLEMENTACIÓN

Una vez definidas la interfaz y funcionalidad del Widget, podemos dar paso a la tercera fase del proyecto, que consistirá en implementar el Widget. En el apartado "Diseño e Implementación en las diferentes plataformas", se cubrieron los aspectos básicos de esta fase de desarrollo para la plataforma Macintosh, los cuales se ampliarán y explicarán con más detalle a continuación.

En esta sección podemos diferenciar dos partes: implementación de la interfaz e implementación de la funcionalidad. Si bien es cierto que el orden de ejecución de las mismas no tiene por qué ser siempre el mismo, incluso se podrían hacer de forma paralela, opté por seguir ese orden durante el desarrollo del widget.

Decidí crear en primer lugar la parte externa del widget, esto es, la interfaz de usuario. El hecho de haber diseñado previamente un prototipo en formato digital facilitó mucho el proceso. Así pues, procedí a obtener todo lo necesario para construir la parte visual del widget. A partir del archivo en formato de capas del prototipo, aislé cada una de las diferentes partes y las guardé de forma independiente en formato de imagen ".png" (Portable Network Graphics).

El siguiente paso era generar un archivo .html donde crear la interfaz propiamente dicha, haciendo uso además de las hojas de estilo CSS para aislar el contenido del formato. Para realizar esta parte, en lugar de utilizar un editor de textos tradicional para cada archivo, hice uso de la aplicación "Dashcode", un entorno de trabajo que facilita la gestión y manipulación de los diferentes archivos necesarios para crear un widget (html, css, javascript, info.plist, imágenes, etc.), descrito con más profundidad en la sección correspondiente

• HTML

El archivo .html lo estructuré a base de capas mediante la etiqueta . Hice dos capas generales, una para la parte frontal y otra para la parte trasera del widget. A su vez, dentro de cada una de las capas, puse nuevas capas conteniendo los diferentes elementos del widget. Concretando, dentro del cuerpo del documento, puse dos capas principales. La primera capa, "front", para la parte delantera contiene a su vez las capas: "front_img", (con la imagen de fondo del widget), "fondos" (a su vez contiene tantas capas como fondos de deportes), "laterales" (contiene cuatro capas, una para cada botón lateral y sus homónimos seleccionados), "cabeceras" (dentro hay tantas capas como deportes, una cabecera por deporte), e "info" (botón de información). La segunda capa principal, "back", contiene todas las capas que se muestran en la parte trasera, que son: "back_img" (la imagen de fondo de la parte trasera), "mensaje" (contiene el mensaje para el usuario), "botones_volver" (contiene una capa para cada botón "volver") y "form" (en esta capa situamos el formulario).

Esquema de la arquitectura de capas utilizada:

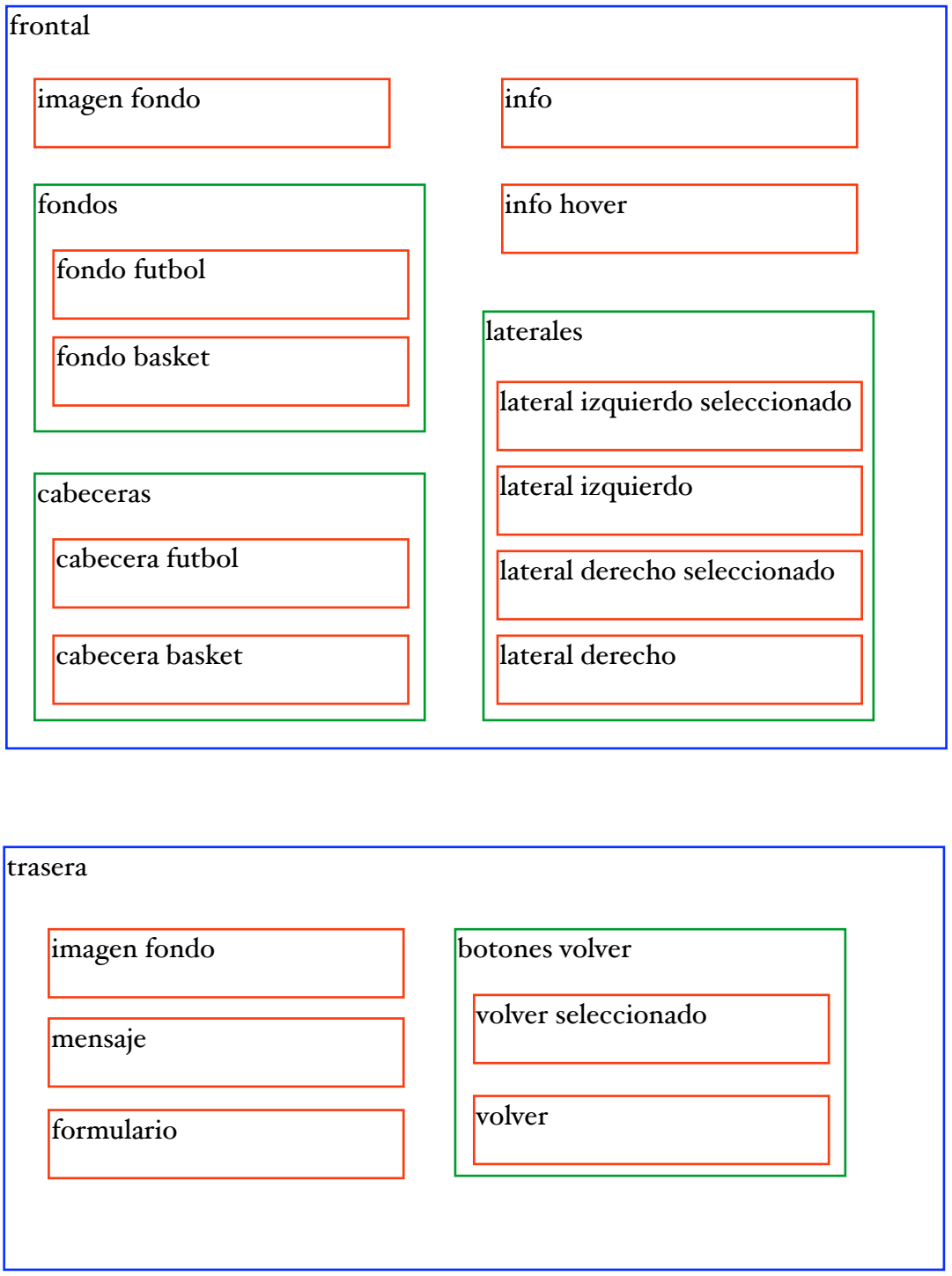


Figura 2.18: Esquema de arquitectura de capas

¿Por qué capas? Lo cierto es que se podrían conseguir los mismos resultados con una estructura de código diferente, por ejemplo, poniendo directamente las referencias a las imágenes. Sin embargo, el trabajar con capas aporta diferentes ventajas. En primer lugar el código es mucho más comprensible y estructurado. Además, cualquier ampliación en el contenido del widget se puede realizar de forma muy sencilla, sencillamente añadiendo los elementos deseados en las capas correspondientes. También resulta muy ventajoso para dar formato, puesto que al dar un identificador a cada capa, podemos acceder a las mismas desde la hoja de estilo CSS con gran facilidad.

• DISEÑO CON CSS

Con el archivo .html ya esquematizado, el siguiente paso fue dar formato a su contenido, para que fuera visualmente atractivo. Esto lo hice mediante la tecnología que proporcionan las hojas de estilo CSS. En vistas a este apartado, en la parte previa doté a cada capa de un nombre identificador, a partir del cual accedemos a cada uno de los elementos desde el archivo .css anteponiendo al identificador el símbolo '#'. El trabajo consistió en ir modificando las propiedades de cada capa hasta que los elementos ocuparan la posición deseada. La mayoría de elementos sufrieron modificaciones en los atributos `width`, `height`, `position`, `left` y `top` (anchura, altura, posición, izquierda y parte superior) con tal de obtener los resultados deseados.

Este no siempre es el método más recomendado (modificar el formato de los elementos según su identificador) al trabajar sobre páginas web convencionales. Sin embargo, en este caso, y para construir la interfaz, resulta apropiado, puesto que trabajamos exclusivamente con imágenes, y resulta necesario especificar las propiedades elemento por elemento. En una fase más avanzada de la implementación, en los apartados de funcionalidad se accederá a los elementos de una manera más eficiente.

En este punto, la interfaz ya está construida, aunque todavía no está lista para ser mostrada, apareciendo muchos de los elementos superpuestos, ya que su visibilidad quedará determinada por el uso del widget. Esto apunta a que el próximo paso es implementar la navegación, y por tanto, gestión de las diferentes capas.

- **PROGRAMACIÓN CON JAVASCRIPT**

El objetivo en este momento era sencillamente hacer funcionar la interfaz, convertir los diferentes imágenes en botones, y repercutir los cambios pertinentes al hacer clic sobre los mismos. Esta parte de implementación, todavía correspondiente a la interfaz, se realizó mediante la tecnología JavaScript, un lenguaje de programación que permite añadir interactividad a los documentos html.

El código JavaScript necesario en este momento se puede incrustar en el archivo .html sin problema alguno. Sin embargo, decidí ponerlo en un archivo aparte, de extensión .js, dedicado exclusivamente a las funciones y código necesarios para ejecutar el widget, con la finalidad de obtener un código más comprensible. De nuevo, el archivo lo generé mediante la aplicación Dashcode, que facilita la gestión y organización de las diferentes partes del widget.

La necesidad más apremiante a la hora de hacer que la interfaz fuera interactiva, era el poder mostrar u ocultar las diferentes capas, según la actividad del usuario. Por ello, el primer paso fue crear dos funciones en JavaScript que permitieran realizar exactamente esta función, pasándoles por parámetro el nombre de la capa. Las funciones son relativamente sencillas y se muestra a continuación:

```
function mostrar(nombreCapa) {

    document.getElementById(nombreCapa).style.display="block";

}

function ocultar(nombreCapa) {

    document.getElementById(nombreCapa).style.display="none";

}
```

Una breve explicación de las funciones:

JavaScript es un lenguaje de programación orientado a prototipos. Uno de sus objetos predeterminados, es el objeto `document`, justo debajo de `window` en la jerarquía DOM. `Document`, contiene todas las imágenes, links, formularios, etc. que aparecen en una página web. Por consiguiente, lo que hacemos en estas funciones es: acceder a la instancia del objeto `document` de nuestro widget, llamar a una de sus funciones para obtener uno de sus objetos a través de un identificador, y cambiar sus propiedades, en este caso, su visibilidad. La cambiamos a `block` para que sea visible y `none` en el caso contrario.

JavaScript permite trabajar con controladores de eventos, definidos en el DOM, y puede detectar cuándo el usuario ha desplazado el cursor sobre una determinada área o hecho clic en cierto elemento. Haciendo uso de estas características, basta con definir en cada imagen cuáles son los eventos ante los cuales se tiene que producir cambios, y qué es lo que tiene que ocurrir cuando se lleven a cabo. La mayoría de botones tendrán definido el evento `onclick`, para desencadenar ciertas acciones ante un clic del usuario, salvo el botón de información que además incorporará un evento `onmouseover`, reaccionando al posicionar el cursor encima.

En primer lugar, implementé la interactividad de los dos botones principales de la parte frontal del widget, "Noticias" y "Resultados". Al hacer clic sobre uno de ellos, se tiene que resaltar, a la vez que el otro botón pasa a tener aspecto de desactivado en caso de que estuviera resaltado. Para hacer esto, basta con mostrar y ocultar cada imagen del botón según hacemos clic sobre las mismas, comportamiento que encapsulé dentro dos funciones: `toggle_left()` y `toggle_right()`, mostradas a continuación:

```
function toggle_left() {        //resaltar botón izquierdo 'Noticias'

    mostrar('lateral_dcho');
    ocultar('lateral_dcho_sel');
    mostrar('lateral_izq_sel');
    ocultar('lateral_izq');

}

function toggle_right() {       //resaltar botón derecho 'Resultados'

    mostrar('lateral_dcho_sel');
    ocultar('lateral_dcho');
    mostrar('lateral_izq');
    ocultar('lateral_izq_sel');

}
```

El concepto envuelto en estas funciones es muy simple. `Toggle_left()` se encargará de resaltar el botón lateral izquierdo, 'Noticias', realizando la pertinente desactivación del botón opuesto (no pueden estar ambos activos al mismo tiempo). El nombre me pareció apropiado puesto que el término del inglés "toggle" representa un cambio de un estado a otro, como es el caso. Al ejecutarse la función, sencillamente se muestra la imagen correspondiente al botón activado, ocultando su homónimo desactivando, y realizando los cambios inversos en el botón opuesto, ocultar la imagen del botón activado y mostrar el desactivado, utilizando las funciones 'ocultar' y 'mostrar'.

Para que las funciones se ejecuten al presionar los botones, modificamos el documento html para incluir los eventos en las imágenes pertinentes:

```
<div id="lateral_izq"> <img src="Images/lateral_izq.png"
onclick="toggle_left();"> </div>

<div id="lateral_dcho"> <img src="Images/lateral_dcho.png"
onclick="toggle_right();"> </div>
```

Como se puede observar basta con incluir en la etiqueta `<img>` el evento `onclick` y a continuación la función de JavaScript que se tiene que ejecutar.

El siguiente paso fue activar el comportamiento del botón de información, para poder acceder a la parte posterior del widget. Este botón, queda por defecto "semi-oculto", para no llamar demasiado la atención. Al pasar el cursor por encima se resalta. Logramos este comportamiento directamente añadiendo un botón perteneciente a la clase `AppleInfoButton`, con las propiedades ya definidas, así como los métodos y constructoras. Así pues basta con añadir una capa para el botón y especificar su construcción en la función `setupParts()` :

```
<div id="infobutton" ></div>
```

Al hacer click, se activará el evento `onclick`, al cual se ha asignado el controlador `showBack(event)`, que es la que realmente se encarga de animar nuestro widget y mostrar la parte trasera. (Notamos que el comportamiento del evento no se define en el documento html, a diferencia del resto de botones, sino que se especifica automáticamente al invocar su constructora, tal y como se explica en la parte correspondiente a las clases Apple del anexo).

El código de la función se muestra a continuación:

```
function showBack(event)
{
    if (window.widget)
        widget.prepareForTransition("ToBack");

    mostrar("front");
    ocultar("back");

    if (window.widget)
        setTimeout('widget.performTransition();', 0);
}
```

Cuando apretamos el botón de información, se invoca esta función, que muestra la parte trasera del widget. Lo primero que hacemos es preparar nuestro widget para darle la vuelta. `widget.prepareForTransition("ToBack")` congela la interfaz mostrada en ese momento, lo que quiere decir que cualquier cambio en la interfaz de usuario del widget después de este punto de ejecución no se mostrará. A continuación, se oculta la capa frontal y mostramos la parte trasera. Finalmente, se ejecuta la transición hacia la parte trasera, dándole la vuelta al widget, quedando la parte delantera congelada y la parte de atrás de la interfaz activa.

Es destacable el hecho de que para ejecutar la acción `performTransition()` (la que ejecuta la animación para voltear el widget) se utiliza la función `SetTimeout()`, con un retardo nulo (o segundos). Esto consigue que la transición se ejecute en la siguiente vuelta del bucle del evento, garantizando que ambas caras del widget han quedado en un estado correcto (ocultadas o activas, según corresponda).

El uso y funcionamiento de `prepareForTransition("ToBack")` y `performTransition()` se detallan en el apartado "El objeto Widget" del Anexo.

El siguiente paso en la implementación de la interfaz fue añadir el formulario a la parte trasera para controlar los cambios producidos al seleccionar un deporte. Introduje el formulario en una nueva capa en el documento html. El código es el siguiente:

```
<div id="form">

<form id="sel_sport" name="selection_form">

<input type="radio" name="sport" value="futbol"checked="checked">Futbol<br>
<input type="radio" name="sport" value="baloncesto">Baloncesto<br>

</form>

</div>
```

Es un simple formulario con dos botones de radio (de momento), uno por deporte, siendo el primero, Fútbol, el seleccionado por defecto. Destacar también que el formulario se inserta en su correspondiente capa, para poder hacer un tratamiento de estilo apropiado y seguir la estructura marcada hasta el momento.

La parte trasera con el formulario queda así:

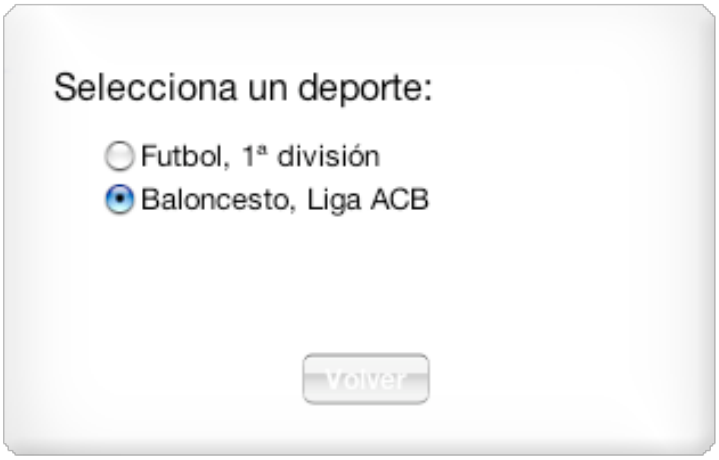


Figura 2.19: Reverso del widget en su versión definitiva

Para volver a la parte frontal, había que seguir un procedimiento muy similar al utilizado para acceder a la trasera: insertar el evento 'onclick' en el botón volver, de forma que al apretarlo se ejecute la función que anima el widget y muestra el anverso.

Como se puede observar en la figura, el botón utilizado "Volver" es diferente al diseñado originalmente en el prototipo. La razón por la que utilicé este nuevo botón en lugar del diseñado originalmente fue que decidí seguir los consejos de la guía de estilo y diseño de Apple. Uno de ellos menciona que la parte trasera de un widget debería usar formularios y botones correspondientes a la interfaz Aqua, interfaz usada en Tiger, puesto que el usuario esperará poder modificar las preferencias de forma rápida y sencilla. Esto será más fácil si observa elementos que le resultan familiares.

Así pues, utilicé la librería de Dashcode para insertar el botón de volver, de acuerdo con las especificaciones de la clase `AppleButton` (se encuentra definida en el glosario). Sólo tuve que asignarle la función `showfront(event)` al evento `onclick`, de forma que al apretarlo se vuelva a mostrar la parte delantera.

Acto seguido, hice los cambios pertinentes en el código para que al seleccionar uno de los deportes y volver a la parte principal, se muestre la repercusión en los elementos: poner el fondo y cabecera correspondientes al deporte elegido. Concretamente, hay que controlar estos aspectos en dos momentos concretos: al pasar de la parte trasera a la delantera, y al mostrar el widget en el dashboard.

Para tal efecto, encapsulé en una función el código a ejecutar para actualizar la parte delantera de acuerdo con la selección del formulario. La función es la siguiente:

```
function updateFront() {

    if (window.document.sel_sport.sport[0].checked == true)
          {
            //si está marcado el radio button futbol
            ocultar('cabecera_basket');
            mostrar('cabecera_futbol');
            ocultar('fondo_basket');
            mostrar('fondo_futbol');

          }

    else if (window.document.sel_sport.sport[1].checked == true)
          {
             //si esta marcado el radio button baloncesto
            ocultar('cabecera_futbol');
            mostrar('cabecera_basket');
            ocultar('fondo_futbol');
            mostrar('fondo_basket');

          }
}
```

Sencillamente comprueba cuál de los botones de radio está seleccionado, y oculta o muestra las capas de la parte delantera en concordancia con la selección. Para que el cambio se haga efectivo, basta con invocar la función en los momentos deseados. Concretamente, desde la función `showFront(event)`, que queda de la siguiente manera:

```
function showFront(event)
{
    if (window.widget)

    widget.prepareForTransition("ToFront");
    mostrar("front");
    ocultar("back");

    updateFront();

    if (window.widget)
        setTimeout('widget.performTransition();', 0);

}
```

Recordamos que también se tiene que actualizar el panel frontal al mostrar el widget en el Dashobard. Para ello, hay que modificar una de las propiedades del objeto widget, onshow, para que cuando detecte que se muestra el widget, realice las acciones deseadas. Definí la función show, y asigné la función a la propiedad:

```
function show()
{
     updateFront();
}

if (window.widget)
{
     widget.onshow = show;
}
```

En lugar de asignar directamente updateFront, (widget.onshow = udpateFront()), le asignamos una función show, dentro de la cual llamamos a updateFront, puesto que es muy posible que más adelante tengamos que hacer más operaciones al mostrar el widget.

La asignación de la función la realizamos dentro de un condicional, puesto que sólo la haremos en caso de que el widget exista. Más adelante también añadiremos más asignaciones a otras propiedades.

Llegados a este punto, la interfaz de usuario ya es navegable completamente, y se comporta de la manera esperada, con la salvedad de que todavía no es funcional. Falta sin embargo un último detalle por implementar, que incluiré en esta etapa, aunque podría también formar parte de la siguiente, que es guardar las preferencias del usuario. Se entiende que al abrir el widget de nuevo después de cerrarlo, sería interesante mostrar el widget tal y como estaba antes. Además, también hay que tener en cuenta que nada impide al usuario cerrar su sesión o reiniciar la máquina mientras el widget está abierto. Cuando el usuario vuelve a su sesión, Dashboard abre el widget automáticamente, y el usuario esperará encontrar el widget en el mismo estado que cuando lo dejó.

Para guardar las preferencias del usuario, Apple proporciona dos métodos:

- `widget.setPreferenceForKey(preference, key)`
- `widget.preferenceForKey(key)`

El primero permite almacenar una preferencia dada `preference` asociada a una clave `key`, si le pasamos el valor `null` como preferencia, se elimina la preferencia enlazada con la llave facilitada. Por otra parte, el segundo método obtiene la preferencia asociada a la clave pasada por parámetro, `key`, o retorna `undefined` si no existe la clave.

Son dos las preferencias que nos interesa guardar en este caso: el último deporte seleccionado, y el botón lateral que se estaba consultando ("Noticias" o "Resultados"). Decidí crear por tanto dos variables globales, `gLateral` y `gRadio`, que servirían para almacenar los valores de las preferencias en el momento de cerrar el widget, y también para insertar los valores obtenidos al iniciar de nuevo el widget. Estas variables, son globales dentro de cada instancia del widget, puesto que nada impide ejecutar varias instancias del mismo widget de forma simultánea. El siguiente paso fue encontrar la manera de guardar en ambas variables un valor representativo mediante el cual identificar qué botones o radios estaban pulsados en el último instante de tiempo. Respecto a los botones laterales, la solución fue modificar las dos funciones ya existentes, `toggle_right` y `toggle_left`, de forma que cada vez que se utilicen, se actualice el valor de `gLateral`. Añadimos al final de cada una respectivamente:

```
    gLateral = "noticias";

    gLateral = "resultados";
```

Para poder saber cuál de los botones de radio estaba pulsando en último lugar, la solución fue crear una función llamada `checkRadios()`, que sencillamente consulta cuál de los botones está pulsado y almacena el número del radio en la variable global `gRadio`.

El siguiente paso fue modificar el método asignado al evento `onremove`, para guardar las preferencias obtenidas:

```
function remove()
{

    checkRadios();

    widget.setPreferenceForKey(gLateral,"lateralSeleccionado");

    widget.setPreferenceForKey(gRadio,"radioSeleccionado");

}
```

Este método se ejecutará justo cuando el widget se cierre, y lo que hace es: consultar cuál de los botones de radio está activado y almacenarlo en la variable global y guardar las dos preferencias, asociadas a sus respectivas llaves para consultarlas al abrir el widget de nuevo.

Ya nos hemos asegurado de que al cerrar el widget, se guardarán los valores deseados. Por consiguiente, ahora se puede restaurar la interfaz de acuerdo con estos valores al mostrar de nuevo el widget. Lo hice de la siguiente manera:

Lo primero fue crear una función `getPreferences()`, que se encargará de rescatar del olvido las preferencias almacenadas al cerrar el widget. Su contenido es el siguiente:

```
function getPreferences()
{
    //Obtenemos el último lateral seleccionado
    var lateralSeleccionado = widget.preferenceForKey
    ("lateralSeleccionado");

    //Obtenermos el último radio seleccionado
    var radioSeleccionado = widget.preferenceForKey("radioSeleccionado");

     // Si había alguno seleccionado
    if (lateralSeleccionado && lateralSeleccionado.length > 0)

        gLateral = lateralSeleccionado;  //obtenemos su identificador

    //caso contrario asignamos identificador por defecto: Noticias
    else
        gLateral = "noticias";

     // Si había algún botón de radio seleccionado
    if (radioSeleccionado && radioSeleccionado.length > 0)

        gRadio = radioSeleccionado;  //obtenemos su identificador

    //caso contrario asignamos identificador por defecto: futbol(0)
    else
        gRadio = "0";

}
```

Obtiene mediante uno de los métodos que proporciona Apple las preferencias asociadas a cada una de las llaves. Está claro que la primera vez que ejecutemos el widget no tendrán ningún valor, por tanto, se tiene que comprobar si contienen la correspondiente cadena de texto. En caso afirmativo la almacenamos en su variable global. En caso contrario, le damos a la variable global un valor por defecto.

Con los valores de las preferencias ya guardados, sólo quedaba hacer una función que permitiera ajustar la interfaz de acuerdo con los mismos. La función, llamada `restoreInterface()` es la siguiente:

```
function restoreInterface()
{

    //Ajustamos la interfaz de acuerdo con las preferencias obtenidas
    if (gLateral == "noticias")
         toggle_left();

    else
        toggle_right();


    window.document.sel_sport.sport[gRadio].checked = true;

}
```

Su comportamiento es muy sencillo y se reduce a una serie de condicionales. En caso de que el último botón lateral apretado fuera el de "noticias", invocamos la función `toggle_left()`. En caso contrario, su homónima, `toggle_right()`. Por último, marcamos el botón de radio que coincide con el almacenado en la variable `gRadio`, cuyo valor se corresponde con el índice del vector que hace referencia al botón.

El último paso fue hacer que estas funciones se ejecuten en el momento correspondiente. Modificamos por tanto las función asignada al evento `onload`, invocando de forma ordenada a cada una de ellas, de forma que al cargar de nuevo el widget, la apariencia sea exactamente la esperada:

```
function load()
{
    setupParts();
    getPreferences();  //Rescatamos las preferencias almacenadas
    restoreInterface();  //Restauramos la interfaz

}
```

En este punto del desarrollo del widget, la interfaz de usuario ya es completamente navegable, y los diferentes cambios que se pueden hacer tienen su correspondiente repercusión en el aspecto visual del widget. Es el momento de añadir funcionalidad al widget. Esta parte de la implementación también la realicé con el lenguaje de programación JavaScript.

Antes de pasar a detallar los pasos que seguí para implementar los diferentes comportamientos del widget, explicaré cuáles son los requisitos funcionales en este caso. El objetivo de nuestro widget es proporcionar al usuario de forma rápida y concisa información relacionada con el mundo del deporte. Concretamente, el usuario puede decidir entre consultar las últimas noticias o los últimos resultados de su deporte favorito.

Si selecciona la pestaña "Noticias", el widget mostrará en su área de contenidos los titulares más recientes en orden cronológico, relacionados con el deporte que en esos momentos esté seleccionado en la parte trasera (por defecto el primero, fútbol). Estos titulares, además, serán enlaces directos a la página web de donde se han extraído, de forma que si el usuario está especialmente interesado en uno de ellos, podrá hacer clic encima. Esta acción hará desaparecer el panel de widgets y abrirá el navegador web por defecto del usuario, cargando la página correspondiente al titular.

En el caso de que el usuario seleccione la pestaña "Resultados", se mostrarán en el área de contenidos los últimos resultados de los acontecimientos deportivos relacionados con el deporte seleccionado, siempre en orden cronológico (de más a menos reciente). Cada línea de resultados, correspondiente al marcador del partido en cuestión, podrá también ser clicada por el usuario, acción que abrirá el navegador web y mostrará una página web con más detalles sobre el partido elegido.

En cualquier momento, el usuario podrá acceder a la parte posterior del widget, haciendo clic en el botón de información situado en la esquina inferior derecha. En este apartado, se permitirá al usuario elegir uno de los diferentes deportes disponibles, acción que quedará consolidada al retornar a la parte anterior mediante el botón "Volver". Al acceder de nuevo a la parte frontal, se actualizará el área de contenidos del widget, de acuerdo con el deporte y la pestaña seleccionados. Además, el usuario podrá pasar de la pestaña "Noticias" a "Resultados" y viceversa sin restricción alguna. A cada vez, se actualizará el área de contenidos para mostrar el material correspondiente a la pestaña elegida.

Si el proceso de actualización del área de contenidos es fallido, por la razón que sea, (la conexión a Internet no está disponible, el servidor de noticias o resultados no está activo, etc;), se insertará en el área de contenidos un breve mensaje comunicando al usuario que no ha sido posible obtener la información solicitada.

Por otra parte, el usuario puede interrumpir el uso del widget en todo momento, mostrándolo, ocultándolo, etc. Cada vez que el widget sea mostrado de nuevo, su estado será idéntico al mostrado en la ocasión anterior. Si es la primera vez que se utiliza, se mostrará el contenido por defecto.

El código necesario para desarrollar esta fase de implementación fue generado en su mayor parte mediante la aplicación Dashcode. Esta aplicación, ofrece, entre otras, la posibilidad de generar automáticamente el código necesario para un tipo de widget concreto. Por consiguiente, basta con crear un nuevo proyecto y elegir una de las plantillas que se ofrecen, en este caso la plantilla para RSS. Esta plantilla crea un widget que muestra los titulares y artículos de una fuente RSS. Para personalizarlo, es tan sencillo como arrastrar la URL del canal RSS deseado en el canvas o lienzo del widget.

La estructura del widget en desarrollo es más compleja, así que resultó necesario realizar varias modificaciones en el código obtenido de la plantilla, para ajustarlo a las necesidades de este widget. Los cambios realizados serán explicados a continuación.

Antes de empezar con el código, tuve que ajustar el área de scroll creada por defecto por la plantilla para ajustarla al área de contenidos del widget. Fue suficiente con modificar los atributos CSS y realizar las modificaciones pertinentes en el documento HTML.

En primer lugar, tuve que encontrar en la red de redes los RSS a partir de los cuales generar los contenidos del widget. En total 4 diferentes canales eran necesarios, dos por deporte (uno para noticias y otro para resultados). La plantilla por defecto proporcionada por Dashcode sólo permite mostrar un único canal RSS, así que tuve que ajustar el código para que se pudieran utilizar tantos como fueran necesarios, en este caso 4. El primer paso fue modificar el archivo `attributes.js`, que contiene los diferentes atributos necesarios para llevar a cabo las operaciones de RSS, y añadir una nueva variable para cada RSS.

El siguiente paso fue modificar la función `getFeedSource()`, encargada de obtener la URL del canal del archivo `attributes.js` y editarlo para convertirlo en una URL válida (retirar símbolos inválidos, quitar espacios, pasar los caracteres a minúsculas, etc.). En este caso, había que determinar cuál de los 4 canales obtener. El proceso fue, comprobar cuál de los deportes está seleccionado y qué pestaña (Noticias o Resultados) se tiene que mostrar. Para ello hice uso de las variables globales `gRadio` y `gLateral`, definidas previamente, las cuales almacenan en todo momento los valores seleccionados.

El código de la plantilla por defecto, muestra en el área de contenido los diferentes titulares del canal RSS en curso, junto con las primeras líneas del artículo, añadiendo al final la fecha. Además, cada diferente artículo se separa con una línea. Este comportamiento no es exactamente el deseado en nuestro caso, puesto que el objetivo del widget era mostrar únicamente los titulares, en orden cronológico. Por consiguiente,

se tuvieron que hacer varios cambios con tal de adaptar los resultados a nuestras necesidades.

Tuve que modificar varias funciones. La primera de ellas, `createRow(title, link, date, description, index)`, encargada de generar una nueva fila de contenido en el área de contenidos del widget. Fue suficiente con eliminar el código que inserta un separador entre cada titular y también la sección de la función encargada de insertar las primeras líneas del artículo. Ahora ya sólo se ven los titulares. Para no mostrar la fecha, en cambio, bastó con modificar uno de los atributos globales, `showdate`, igualándolo a 0.

Tras conseguir que los contenidos se mostraran de la manera deseada, el siguiente paso fue hacer que se mostraran los diferentes resultados en el momento oportuno. Como en este punto del desarrollo la interfaz ya estaba completa, sencillamente tuve que insertar el código precisado en las diferentes funciones ya creadas previamente. La única tarea a realizar era, básicamente, actualizar el canal RSS mostrado en el momento oportuno. Encapsulé la funcionalidad en un único método, `updateFeed()`, con los siguientes contenidos:

```
function updateFeed()
{
    htmlFeedURL = getFeedSource();
    feed.url = htmlFeedURL;
    refreshFeed();

}
```

La función determina cuál de los canales RSS tiene que ser mostrado, y después invoca a `refreshFeed()`, función generada por Dashcode que inicializa el proceso de obtención de datos de los servidores. Sencillamente tuve que incluir la función en los puntos necesarios, o sea, en la función encargada de mostrar la parte frontal `showFront(event)`, y en las funciones que cambian la selección de los laterales, de Noticias a Resultados y viceversa, `toggle_left()` y `toggle_right()` respectivamente.

Antes de pasar a la siguiente fase de desarrollo, explicaré la estructura que sigue el código utilizado para obtener los titulares de los canales RSS y mostrarlos en el widget.

El proceso se desencadena cuando el widget detecta que ha sido mostrado, mediante el evento `show()`. Previamente se tiene que haber cargado, lo cual activa el evento `onload()`, que invoca a la función `load()`. En este momento, se determina cómo se tiene que mostrar la interfaz, y cuál de los canales RSS se utilizará. Cuando el widget se muestra, se invoca la función `refreshFeed()`, que inicializa el proceso de conexión con el servidor.

Esta función, es la que crea e inicializa el objeto `XMLHttpRequest` (descrito en la parte correspondiente del anexo), que nos permite contactar con el servidor y obtener la información deseada. Se inicia una operación de tipo "`GET`", puesto que queremos solicitar datos.

```
function refreshFeed()
{
    showLoadingMessage();
    if (xml_request != null) {
          xml_request.abort();
          xml_request = null;
     }
    xml_request = new XMLHttpRequest();

    xml_request.onload = function(e) {finishedLoadingXML(e, xml_request);}
    xml_request.overrideMimeType("text/xml");
    xml_request.open("GET", feed.url);
    xml_request.setRequestHeader("Cache-Control", "no-cache");
    xml_request.send(null);
}
```

Destacamos que cuando el objeto `XMLHttpRequest` ya se ha cargado, se invoca la función `finishedLoadingXML(e, xml_request)`, donde se pasan a realizar las operaciones de procesado de datos. Concretamente, se accede a los resultados mediante las propiedades del objeto creado y se procesan haciendo uso de los métodos del DOM. Una vez obtenidos los fragmentos deseados, se insertan en el área de contenidos mediante el método `addEntriesToContents(contents, results)`. Esto se hace modificando el código HTML de nuestro widget a través de funciones como `document.createElement('div')`, los cual permite crear de forma dinámica el área de contenidos en base a los resultados extraídos de los servidores.

Todo este proceso de obtención de datos de los servidores no se repite cada vez que se muestra el widget, ya que podría resultar un uso excesivo de los recursos de la máquina del usuario. Sólo será necesario repetirlo para refrescar los titulares de noticias y resultados. Esto se realiza de forma periódica a intervalos de, como mínimo, 15 minutos. Cada vez que se muestra el widget, se comprueba cuánto tiempo ha pasado desde la última vez que se refrescaron los contenidos, y se inicia el proceso de refresco en caso de que supere los 15 minutos.

```
function show()
{
   updateFront();  //Actualizamos la parte frontal
   var now = (new Date).getTime();
   // sólo refrescamos los titulares si han pasado 15 minutos
   if ((now - last_updated) > 900000) {
      refreshFeed();
   }
}
```

El widget ya está casi listo para ser utilizado. De hecho, ya funciona del todo. El único paso que queda pendiente por hacer es diseñar un icono para el widget, que será mostrado en la lista de widgets para representarlo.

Para acabar el widget es necesario incluir un icono en miniatura del widget. Para realizarlo, hice uso de las mismas aplicaciones de edición gráfica utilizadas previamente para diseñar la interfaz del widget. En este caso, las medidas y forma del icono están mucho más estandarizadas, así que me limité a establecer un fondo con los ajustes por defecto (en lo referente a tamaño y figura). Para seguir un estilo acorde con el widget, decidí incluir en el icono las imágenes correspondientes a los deportes utilizados, en este caso un balón de fútbol y de baloncesto. Así pues, el fondo consistió en un degradado de tipo reflectante entre los colores blanco y negro, quedando en el centro una franja de color blanco. Situé medio balón en cada lado del widget, dejando espacio en el centro para colocar el título deportes mediante una capa de texto. El resultado fue el siguiente:

Figura 2.20: Icono del widget en el Dashboard

A continuación unas capturas del widget en funcionamiento:

Figura 2.21: Capturas del widget en funcionamiento

En este punto el widget ya está acabado. Han finalizado todas las etapas de creación. El siguiente paso es incorporar el widget al Dashboard para poder utilizarlo como cualquier otro widget. La manera de hacerlo es muy sencilla. Una opción es colocar todos los archivos en una misma carpeta y añadirle la extensión .wdgt. El sistema lo reconoce de inmediato como widget y sustituye el icono de la carpeta por el icono por defecto de los widgets. Un doble clic sobre el icono bastará para instalar el widget en nuestro Dashboard.

Figura 2.22: Icono por defecto de los widgets.

Otra opción es instalar el widget en el Dashboard directamente a través de Dashcode mediante una de las opciones de sus menús: "Deploy Widget to Dashboard...", lo cual facilita aún más si cabe el proceso.

Si el widget es de nuestro agrado y creemos que puede resultar de utilidad a la comunidad de usuarios de widgets, también podemos subirlo a la red, a través del siguiente enlace:

http://www.apple.com/downloads/dashboard/?r=dbw

Diseño de Widgets

Ya hemos cubierto las diferentes formas y métodos utilizados para implementar un Widget. Igual de importante es la interfaz gráfica del widget. En esta sección se darán algunos consejos útiles y principios a tener en cuenta a la hora de diseñar un widget. Buena parte de esta información ha sido extraída de la documentación para desarrolladores que facilita Apple, (concretamente del documento: Dashboard Programming Topics > Designing Widgets, 2006), aunque es aplicable en general para widgets operativos sobre cualquier plataforma.

Guías de diseño para la interfaz principal

La parte más importante del widget, es la que se encuentra en la cara frontal (las preferencias se suelen mostrar en la parte trasera). Es donde se producen la mayoría de interacciones con el usuario. A la hora de diseñar la cara frontal, es recomendable tener en cuenta los siguiente consejos:

- El diseño del widget debería centrarse en transmitir inmediatamente su propósito principal.
- Hacer un uso eficiente del espacio, tratando de mostrar únicamente información útil.
- Mostrar la información de forma inmediata. El Dashboard se muestra y oculta muy rápidamente, a voluntad del usuario, con lo que obligarlo a esperar a que se muestre el contenido puede resultar frustrante y una pérdida de tiempo.
- Diseñar el widget para que tenga una funcionalidad bien diferenciada. No debería necesitar explicación o configuración alguna. En lugar de crear un widget que cumple 3 diferentes funciones, podría ser preferible hacer 3 widgets diferentes, de forma que cada uno cumpla una de ellas. Esto provocaría que cada tarea sea diferente a las demás y permite al usuario elegir la que le resulta más útil.

Figura 2.23: Ejemplo de widget sobrecargado, el que mucho abarca, poco aprieta

Figura 2.24: 3 widgets sencillos mejor que 1 complejo. Cada uno centrado en una sola tarea

- Se debería tener en cuenta a los usuarios con pantallas pequeñas. Un iBook tiene una resolución de 1024 x 768 pixeles, así que nuestro widget debería ser un "buen ciudadano" y dejar sitio para otros widgets. Los usuarios suelen tener múltiples widgets abiertos al mismo tiempo, no se debería monopolizar el espacio disponible en pantalla. De lo contrario, puede que el widget no se use por tener un tamaño poco práctico.

Figura 2.25: Un widget grande monopoliza el valioso espacio de la pantalla

Figura 2.26: Un widget pequeño sigue cumpliendo su función y deja sitio para otros widgets

- Usar las barras de scroll con poca frecuencia. El conjunto de información que muestra el widget por defecto debería ser el mínimo posible y no requerir scroll. El uso de una barra de scroll puede ser aconsejable si el widget tiene que mostrar mucha información (e.g. un diccionario), ya que permite reducir el tamaño del widget. Una posibilidad sería incluir en las preferencias una opción que permita mostrar una versión simplificada que no use barra de scroll.
- Utilizar un color que distinga el widget. Un estilo de colores único para el widget permite que el usuario lo reconozca rápidamente cuando lo quiere usar.
- Evitar los colores demasiado llamativos o estridentes. Colores muy contrastados pueden resultar ofensivos para los usuarios. Aunque a veces puede resultar apropiado usar colores diferentes, suele ser mejor mantener los colores en una misma gama, para que el widget sea más atractivo a la vista.

Figura 2.27: Ejemplo de widget ofensivo

- Usar fuentes de texto claras y legibles. Los usuarios esperaran poder obtener información del widget rápidamente. Evitar sacrificar la legibilidad por conseguir una apariencia o estilo determinados.

- Evitar la publicidad en la parte frontal del widget. Identificar el widget por la marca es aceptable, pero la publicidad desaprovecha el espacio del widget. La presencia del widget en el Dashboard del usuario es un privilegio. Es mejor usar la parte trasera para la información no vital en el propósito del widget, como marca, licencias, copyright, etc.

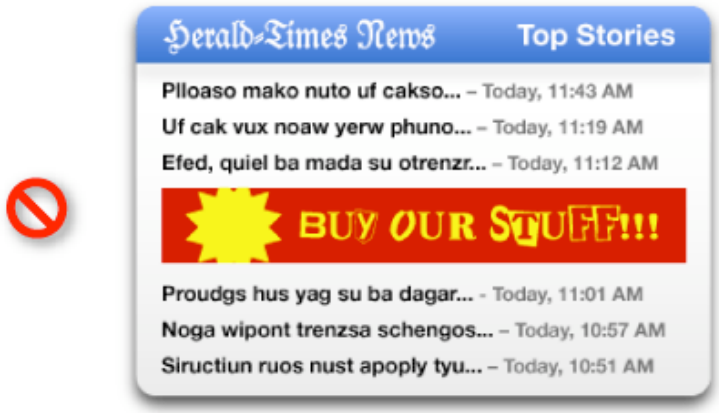


Figura 2.28: No malgastar espacio con anuncios publicitarios

Figura 2.29: La información no vital puede ir en la parte trasera

- Usar los identificadores `CloseBoxInsetX` y `CloseBoxInsetY` del archivo Info.plist para situar la caja de cierre en la esquina superior izquierda del widget. Debería estar posicionada encima del widget, para que se vea la relación entre la caja de cierre y el widget.
- Dar soporte a operaciones de copiar y pegar cuando sea posible. Muchos usuarios esperan poder copiar y pegar elementos entre aplicaciones y esperarán poder hacer lo mismo con los widgets.

Guías de diseño para el reverso

Si nuestro widget requiere configuración, se pueden mostrar las preferencias en el reverso del mismo. A continuación algunos consejos para diseñar la parte trasera:

- Usar el gráfico del botón de información para dar a entender que se puede acceder a la parte posterior del widget, sea para modificar preferencias o consultar información. El botón de información consiste en una "i" dentro de una circulo que aparece cuando el cursor pasa por encima. Un clic sobre el botón activa una animación del widget para voltearlo. El botón de información es un estándar usado en la mayoría de widgets, y por tanto, los usuarios saben inmediatamente qué quiere decir y qué ocurrirá al apretarlo.

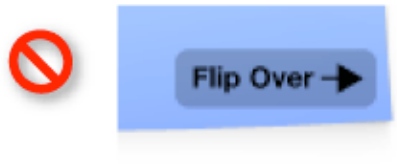


Figura 2.30: Control no estándar para mostrar el reverso

Figura 2.31: El botón estándar - conocido por el usuario

- Utilizar un color más oscuro o por lo menos con un fondo diferente para la parte trasera. Reutilizar el color de fondo de la interfaz principal no es aconsejable ya que puede llevar a confusión sobre cuál es la cara principal.

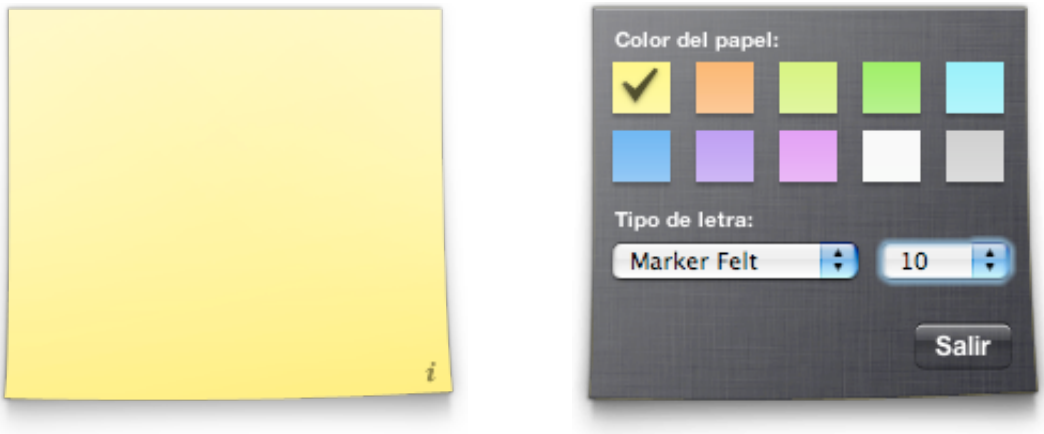


Figura 2.32: Fondos diferentes permiten diferenciar el anverso del reverso

- Situar el botón de información en la esquina inferior derecha del widget, siempre y cuando sea posible. Se puede poner en otras esquinas, pero la esquina inferior derecha es el lugar donde la mayoría de usuarios esperan encontrar el botón.
- Usar la animación de volteo del widget sólo para mostrar la parte trasera. Un uso excesivo de la animación le quitaría seriedad.
- Incluir un botón de "Hecho" o "Volver". Cuando el usuario haya acabado de ajustar las preferencias, al apretar el botón el widget debería darse la vuelta y mostrar de nuevo la parte frontal.
- Si es necesario, mostrar la información de licencias, logos, y ayuda mínima en la parte trasera. Al igual que en la interfaz principal, también se deberían evitar los anuncios.

Figura 2.33: La parte trasera puede ser usada para mostrar una marca o logotipo

III - BLOQUE DE FINALIZACIÓN

Conclusiones

Los widgets se han extendido y son cada vez más populares entre los usuarios. La versatilidad de las tecnologías sobre que se basan ha propiciado una gran expansión de los mismos, dando lugar a un extenso abanico de diferentes widgets. Son varias las compañías que ofrecen widgets, y existe una elevada competencia entre ellas por atraer a los usuarios. Cada compañía trata de diferenciarse de las demás añadiendo características únicas, tanto a nivel de estilo como de funcionalidad, sin embargo, ha quedado patente que las diferencias reales entre las plataformas se reducen a ciertas peculiaridades a nivel de implementación, puesto que el resultado es idéntico en la mayoría de casos. En última instancia, esto redunda en el beneficio de los usuarios finales, puesto que disponen de un amplísimo catálogo donde elegir.

Es cierto también que si bien los usuarios de aquellos sistemas que integran los widgets entre sus funcionalidad suelen utilizarlos con frecuencia, sin recurrir a otras alternativas, son muchos los que deciden no usarlos, a pesar de tener a su alcance todos los recursos necesarios para ellos. Esto apunta a que es necesario minimizar el esfuerzo requerido por parte de los usuarios para usar los widgets si se quiere hacerlos llegar al gran público. La no-esencialidad de los mismos conlleva el hecho de que sólo serán usados si la instalación y configuración no suponen demasiada dedicación.

Por otra parte, de cara al desarrollador, los widgets ofrecen muchas facilidades para su implementación, puesto que se basan en tecnologías tan populares como conocidas (Html, JavaScript, CSS, ...). Esto redunda en claras ventajas, puesto que el soporte y documentación que se pueden encontrar sobre el tema son muy elevados, y ya existen herramientas que facilitan aún más si cabe la generación del código necesario (Dashcode es la más destacable, aunque se podrían usar también otras como por ejemplo Dreamweaver). Sin embargo, es también interesante destacar que la facilidad de creación de los widgets puede llegar a resultar perjudicial. El hecho de que el desarrollar un widget esté al alcance de cualquiera, si bien fomenta la participación de los usuarios, resulta en una avalancha de widgets, muchos de ellos de dudosa utilidad y con bajos niveles de calidad en lo que a funcionamiento y eficiencia se refiere.

A lo largo del trabajo, se ha proporcionado información detallada sobre el concepto y esencia de los widgets. Sin embargo, el proceso de implementación ha demostrado que existe el riesgo para los desarrolladores de caer en la tentación de incorporar demasiada funcionalidad en sus widgets. No hay que olvidar qué es un widget realmente y cuál es su propósito, con el fin de no de desvirtuar su esencia.

La experiencia de realizar un trabajo de investigación y desarrollo sobre plataforma Mac ha resultado plenamente satisfactoria. Existe una gran cantidad de documentación, soporte y toda clase de herramientas, así como ejemplos para consultar, haciendo más fácil la vida del desarrollador. Recomiendo la plataforma sin lugar alguno para dudas,

tanto por su fiabilidad como por sus elegancia, así como por su quasi- inagotable fuente de recursos.

La aplicación Dashboard, como ejemplo de servidor de Widgets, es ejemplar en todos los sentidos. Siempre a una distancia mínima, pone al alcance del usuario sus widgets favoritos, con una presentación y eficiencia destacables. Es la plataforma sobre la que he desarrollado todas las pruebas, y sobre la que de hecho utilizo los widgets de forma regular. Su integración con Dashcode es muy útil y me ha permitido simplificar en gran medida el proceso de creación del widget de este trabajo. Son las herramientas perfectas para crear widgets, y no me queda más remedio que recomendarlas abiertamente a todo aquel que desee probarlas.

Otra apreciación interesante, obtenida como resultado del trabajo, es que la creación de widgets puede tener unos objetivos muy variados. El esfuerzo envuelto no es muy elevado, (dependiendo por supuesto de las características del widget), y por tanto, es viable la creación de widgets para un uso puramente personal, ajustándolo a nuestras propias necesidades. Por otra parte, el campo de los widgets no resulta muy adecuado para la comercialización u obtención de beneficios. La inmensa mayoría de widgets que hay disponibles actualmente son gratuitos, con lo que crear un widget por el que se tuviera que pagar resultaría en un fracaso casi seguro. He podido observar que muchos widgets se limitan a ofrecer la opción de hacer una donación a los creadores (via PayPal). En cualquier caso, no se pueden considerar como una fuente de negocio (aunque lo son de forma indirecta para las compañías que los propulsan).

En definitiva, a día de hoy los widgets están al alcance de todos. A pesar de que su uso todavía no se ha generalizado, se están popularizando cada vez más y veremos cómo poco a poco llegarán a formar parte de la mayoría de dispositivos y sistemas, puesto que han demostrado resultar de gran utilidad.

Trabajo Futuro

Son varios los aspectos que se han quedado en el tintero y que se podrían incluir en el trabajo. También hay algunas características que se podrían mejorar, comentamos ambos a continuación.

En lo que al widget desarrollado se refiere, salta a la vista que una de las ampliaciones más inmediatas de una futura versión sería incluir un mayor número de deportes a elegir (tenis, F1, golf, etc.), para contentar a un mayor número de usuarios. También sería interesante mejorar la visualización de los resultados, dando formato a la información e incorporarla en una tabla o similar, para que fuera mas familiar al usuario. Por otra parte, el código utilizado aceptaría de buen grado varias modificaciones para facilitar tanto su comprensión como eficiencia. Un punto interesante que sin lugar a dudas incluiría sería la conversión del widget a otros formatos para hacerlo compatible con el resto de plataformas. Sería una buena manera de ver si realmente las diferencias son o no significativas. El espectro de usuarios cubierto aumentaría significativamente.

Se podría contemplar también la posibilidad incorporar internacionalización al widget desarrollado. En un principio, el público al que está dirigido parece lo suficientemente acotado como para incluir un único idioma, aunque no se puede descartar que usuarios de otras nacionalidad demostrarán interés por el widget, con lo que podría ser interesante localizarlo para diferentes idiomas. El coste envuelto en incorporar este apartado dependería, no sólo del número de idiomas que se quisieran cubrir, sino también de la disponibilidad de canales RSS en cada uno de los idiomas para los diferentes deportes. En lo referente a los menús del widgets, la tarea sería mucho más sencilla. La propia aplicación Dashcode ofrece la posibilidad de localizar los textos. En el caso de los textos en formato de imagen bastaría con crear sus correspondientes versiones en cada idioma.

A nivel de investigación en el trabajo, se podrían realizar encuestas y cuestionarios más rigurosos con el fin de determinar qué porcentaje de usuarios usa los widgets con frecuencia, cuál es la plataforma preferida, etc. También se podría tratar de discernir qué widgets son los más utilizados en general y cuántos widgets activos de media tienen los usuarios. Estos datos, quizás de un carácter mas bien estadístico permitirían acotar mucho más el estado de los widgets y determinar qué aspectos se podrían pulir. También aportaría riqueza la implementación de un widget con temática diferente, haciendo uso de otras de las tecnologías que pueden participar en la creación widgets y comparar los procesos de desarrollo y la dificultad envuelta en cada uno.

En lo que al mundo de los widgets se refiere, está también claro que se va a seguir avanzando. El sucesor de Mac OS X Tiger, Leopard, llevará el concepto de widgets un paso más allá. Dashboard tendrá integración total con Safari (el navegador web de Apple), y se incluirá una opción que permitirá crear un widget de forma instantánea a partir de cualquier web. Bastará con seleccionar la porción de la página que deseamos convertir en widget, y se trasladará a nuestro Dashboard, manteniendo su

comportamiento dinámico. El resto de compañías también están avanzando en este campo. Por poner un ejemplo, Yahoo! acaba de lanzar la cuarta versión de su servidor de Widgets, Konfabulator 4. Entre sus nuevas características destacan un claro incremento en el rendimiento y eficacia, minimizando el consumo de memoria, y la opción de insertar fragmentos de webs en el escritorio a modo de widgets, de manera similar a Leopard.

También se está avanzando en el campo de widgets móviles. El ejemplo más claro es el reciente lanzamiento de iPhone, el teléfono móvil de Apple, que incluye entre sus funciones varios widgets. Todo apunta a que con el tiempo los widgets formarán parte de la mayoría de dispositivos móviles en un futuro muy cercano.

IV - BIBLIOGRAFÍA

- Dave Thau. (2007). *The Book of JavaScript: a practical guide to interactive web pages*. No Starch Press, Inc.
- Tiger Developer Overview Series. Developing Dashboard Widgets. (2007). Disponible en: http://developer.apple.com/macosx/dashboard.html
- Developer Connection. Introduction to Dashboard Tutorial. (2006). Disponible en: http://developer.apple.com/documentation/AppleApplications/Conceptual/Dashboard_Tutorial/index.html
- Developer Connection. Introduction to Dashboard Reference. (2006). Disponible en: http://developer.apple.com/documentation/AppleApplications/Reference/Dashboard_Ref/index.html
- Developer Connection. Introduction to Safari HTML Reference.(2006).Disponible en:http://developer.apple.com/documentation/AppleApplications/Reference/SafariHTMLRef/index.html#//apple_ref/doc/uid/TP40002049
- Developer Connection. Introduction to Safari CSS Reference. (2007). Disponible en:http://developer.apple.com/documentation/AppleApplications/Reference/SafariCSSRef/index.html#//apple_ref/doc/uid/TP40002050
- Developer Connection. Safari JavaScript Reference. (2005). Disponible en:http://developer.apple.com/documentation/AppleApplications/Reference/SafariJSRef/Intro/Intro.html#//apple_ref/doc/uid/TP40001482-CH2g-BAJDAJAG
- Developer Connection. Introduction to Dashboard Programming Topics. (2006). Disponible en: http://developer.apple.com/documentation/AppleApplications/Conceptual/Dashboard_ProgTopics/index.html#//apple_ref/doc/uid/TP40002837
- Dashboard Widgets. Behind the Dashboard - You're in control.(2005). Disponible en: http://www.dashboardwidgets.com/news/comments.php?id=44_0_2_0_C
- Programación en JavaScript. (2007). Disponible en: http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_JavaScript
- Techype: Proof Mobile AJAX Only Works In A Parallel Reality. (2007). Disponible en: http://techype.blogspot.com/2007/05/proof-mobile-ajax-only-works-in.html
- Innovation Cloud. Widget Week part 1: Mobile Widgets (2007). Disponible en: http://innovationeye.wordpress.com/2007/05/20/widget-week-part-1-mobile-monday-mobile-widgets/
- Opera Mobile. Create mobile widgets/AJAX applications with the Opera Platform SDK - by Eskil Silvertsen. (2005). Disponible en: http://my.opera.com/eskils/blog/show.dml/74871

- TomSoft. Mobile Widgets. (2007) .Disponible en: http://blog.landspurg.net/?cat=14
- TomSoft. What is a (mobile) Widget. (2006). Disponible en: http://blog.landspurg.net/what-is-a-mobile-widget
- X.4 Dashboard (Quarter Life Crisis).(2005). Disponible en: http://earthlingsoft.net/ssp/blog/2005/05/x4_dashboard
- Widget - Wikipedia, la enciclopedia libre. (2007). Disponible en: http://es.wikipedia.org/w/index.php?title=Widget&redirect=no
- Gadget - Wikipedia, la enciploedia libre. (2007). Disponible en: http://es.wikipedia.org/w/index.php?title=Gadget&redirect=no
- Yahoo! Widgets Blog - The Ever-Changing Widget Landscape. (2007). Disponible en: http://widgets.yahoo.net/blog/?p=16
- Yahoo! Widgets Information (2007). Disponible en: http://widgets.yahoo.com/info/
- Yahoo! Widgets FAQ (2007). Disponible en: http://widgets.yahoo.com/faq/
- Yahoo! Widget Engine. Widget Creation Tutorial. (2007). Disponible en: http://widgets.yahoo.com/workshop/
- Google Gadgets API Developer Guide (2007). Disponible en: http://www.google.com/apis/gadgets/docs-home.html
- Microsoft Developer Network. Gadget Development Overview. (2007). Disponible en: http://msdn2.microsoft.com/en-us/library/ms723694.aspx
- Opera Widgets Developers Documentation (2007). Disponible en: http://my.opera.com/community/dev/widgets/
- aDesklets 0.6.1 (2006). Disponible en: http://adesklets.sourceforge.net/doc/en/

V - ANEXO

El objeto Widget

El objeto Widget es un objeto JavaScript que proporciona extensiones especificas para Dashboard. Cuando el widget se carga, el Dashboard crea de forma automática una instancia de este objeto, que podemos usar en el código JavaScript de nuestro widget. El nombre de la instancia por defecto es `widget`.

Propiedades:

- **identifier**

Contiene un identificador único para esa instancia del widget

```
widget.identifier
```

Es una propiedad de lectura, almacena una cadena de texto con un valor único entre todas las instancias de un widget concreto. Este valor lo asigna el Dashboard y persiste entre instanciaciones de cada instancia del widget.

- **ondragstart**

Contiene el controlador de evento a invocar cuando se inicia el arrastre de un widget

```
widget.ondragstart
```

Para ser notificados cuando el widget ha iniciado un "arrastre" o desplazamiento, asignamos una función a esta propiedad. Mediante la función podemos cambiar la apariencia de la interfaz de usuario del widget mientras está siendo arrastrado. La declaración de la función sería del estilo:

```
function ControladorInicioArrastre() { ... }
```

- **ondragend**

Contiene el controlador de evento a invocar cuando finaliza el arrastre de un widget

```
widget.ondragend
```

Para ser notificados cuando el widget ha finalizado un arrastre, asignamos una función a esta propiedad, la cual se ejecutará cuando ocurra el evento. Mediante la función podemos cambiar la apariencia de la interfaz de usuario del widget tras haber sido arrastrado. La declaración de la función sería del estilo:

```
function ControladorFinArrastre() { ... }
```

- **onhide**

Contiene el controlador de evento a invocar cuando se oculta el Dashboard

```
widget.onhide
```

Si queremos detectar cuándo el widget ha sido ocultado, asignamos una función a esta propiedad, para desactivar el widget y dejarlo en estado de hibernación, sin consumir recursos. La declaración de la función sería del estilo:

```
function ControladorOcultacion() { ... }
```

- **onremove**

Contiene la función que se invocará cuando el widget sea eliminado de la capa del Dashboard.

```
widget.onremove
```

Asignamos una función a esta propiedad si queremos realizar alguna acción cuando el widget se elimine del Dashboard. Generalmente se realizarán operaciones como salvar las preferencias, eliminar archivos cache o liberar recursos retenidos. La declaración sería del estilo:

```
function ControladorEliminacion() { ... }
```

- **onshow**

Contiene la función que se invocará cuando se muestra la capa de widget del Dashboard

```
widget.onshow
```

Asignamos una función a esta propiedad si queremos realizar alguna acción cuando el widget ha sido mostrado en el Dashboard. Usaríamos esta función para activar el widget y empezar a procesar información tras suspensión. La declaración sería del estilo:

```
function ControladorMostrarWidget() { ... }
```

Métodos:

- **openApplication**

Lanza la aplicación con el identificador de paquete especificado.

```
widget.openApplication(bundleId)
```

La invocación de este método conlleva la ocultación de la capa del Dashboard.

- **openURL**

Abre la URL especificada en el navegador por defecto del usuario.

```
widget.openURL(url)
```

La invocación de este método oculta el Dashboard, abre el navegador por defecto en caso de que no estuviera abierto e introduce la URL en la barra de navegación. El método no permite abrir URLs que usan el esquema file: a menos que se especifique mediante la llave `AllowFileAccessOutsideOfWidget` en el archivo de propiedades, info.plist

- **preferenceForKey**

Retorna la preferencia asociada con la llave especificada.

```
widget.preferenceForKey(key)
```

Mediante este método podemos acceder al valor de una preferencia almacenada previamente con una llamada a `setPreferenceForKey` valor. Retorna una cadena de texto con los contenidos de la cadena o `undefined` en el caso de que no exista la preferencia.

- **prepareForTransition**

Notifica al Dashboard que se va a producir una transición desde o hacia la parte trasera.

```
widget.prepareForTransition(transition)
```

El método prepara el widget para mostrar u ocultar el reverso. El parámetro puede tomar el valor "ToFront", lo cual deshabilita las actualizaciones de la pantalla en la interfaz del widget con tal de hacer los preparativos para mostrar la parte de atrás, o bien "ToBack", que congela la interfaz con tal de prepararla para mostrar de nuevo los contenidos principales. Se invoca `performTransition` cuando las capas HTML están listas.

- **performTransition**

Ejecuta una animación para pasar de la parte frontal a la posterior y viceversa.

```
widget.performTransition()
```

Este método se invoca después de `prepareForTransition`, para indicar si vamos a mostrar el anverso o el reverso. Cuando se ejecuta el Dashboard inicia una animación para dar la impresión de que el widget se da la vuelta para mostrar otros contenidos.

En la parte previa a la invocación de este método, se tiene que ajustar la hoja de estilo asociada al HTML para reflejar los cambios que se van a mostrar.

- **setCloseBoxOffset**

Cambia la localización del botón para cerrar el widget.

```
widget.setCloseBoxOffset(x, y)
```

Este método desplaza el boton de cierre `x` pixeles desde la esquina superior izquierda del widget e `y` pixeles desde la parte superior del widget. Sólo se aceptan valores comprendidos entre 0 y 100.

- **setPreferenceForKey**

Asocia una preferencia con llave dada.

```
widget.setPreferenceForKey(preference, key)
```

Los parámetros `preference` y `key` contienen cadenas de texto que representan la preferencia que queremos almacenar y la llave identificador con la cual asociarla. Si `preference = null`, se elimina la llave especificada de las preferencias.

Las preferencias son guardadas como texto plano y por consiguiente no se recomienda almacenar contraseñas u otra información delicada mediante este sistema.

- **system**

Ejecuta un comando de la línea de comandos.

```
widget.system(command, endHandler)
```

El parámetro `command` es una cadena de texto con el comando a ejecutar. Debe especificar la ruta completa o relativo a la utilidad del comando de líneas e incluir los argumentos necesarios. El parámetro `endHandler` especifica el controlador a invocar al finalizar la ejecución del comando. Si Null, el método se ejecuta de forma síncrona (el widget se detiene hasta que finaliza el comando)

Introducción y uso de las clases de JavaScript para Apple

A partir de la versión 10.4.3 de Mac OS X, Apple facilita una serie de clases de JavaScript que facilitan la inclusión de controles y utilidades de uso común en los widgets. Estas clases (Apple Classes), incluyen:

- Áreas y barras de scroll
- Deslizadores
- Temporizadores de animación
- Botones
- El botón Info

Las clases se encuentran por defecto en la siguiente ruta: `/System/Library/WidgetResources/AppleClasses/` Se pueden utilizar o bien indicando tal ruta o incluyéndolas directamente en el widget, en el caso de que queramos que nuestro widget sea retrocompatible con versiones más antiguas del sistema operativo. En el proyecto que nos ocupa, decidí incluir toda la información necesaria en el paquete del widget, permitiendo su uso en versiones anteriores.

Para utilizar las clases, añadimos el controlador `load()` para el evento `onload`. Dentro del controlador insertamos la función `setupParts()`, que se encargará de crear los diferentes objetos de cada clase utilizada al cargar el widget.

El uso de estas clases se traducen en un importante ahorro temporal a la hora de construir el widget, y permiten además dotarlo de un estilo no sólo atractivo sino también común en la mayoría de widgets, lo que facilita la tarea al usuario, que puede identificar y utilizar tales elementos de una forma estándar. (e.g. el usuario sabrá que el botón de información se encuentra en la esquina inferior derecha y que al apretarlo se mostrará la parte trasera del widget).

A continuación se describe el uso de las diferentes clases utilizadas para implementar el widget.

El botón info

Para insertar un botón de información común a los widgets de apple, tenemos que incluir en nuestro documento HTML las clases correspondientes, que son: `AppleAnimator` y `AppleInfoButton`:

```
<script type="text/javascript" src="AppleClasses/AppleAnimator.js"
charset="utf-8"></script>

<script type="text/javascript" src="AppleClasses/AppleInfoButton.js"
charset="utf-8"></script>

<script type="text/javascript" src="Parts/InfoButton.js" charset="utf-8"></
script>
```

Ahora ya podemos usarlas a discreción, en este caso para poder insertar el botón de información en nuestro widget. Sencillamente incluimos en la función `setupParts()` una llamada a la operación constructora de alto nivel de nuestro botón, de la siguiente forma:

```
CreateInfoButton('infobutton', {foregroundStyle:'white', frontID:'front',
onclick: 'showBack', backgroundStyle: 'black' });
```

Le pasamos como parámetro un identificador para el botón, y una lista con las preferencias: cuál es el identificador de la capa frontal, la función a ejecutar al apretar el botón y el color de la parte delantera y trasera. Esta función se encuentra en el archivo aparte `InfoButton.js`, y nos permite usar la constructora de la clase con un mayor nivel de abstracción.

Incluimos el botón en nuestro documento html, a modo de capa con el mismo identificador especificado en la constructora:

```
<div id="infobutton" ></div>
```

Ya podemos editar su formato desde la hoja de estilo CSS accediendo al identificador de la capa.

AREAS DE SCROLL

Apple también proporciona una serie de clases JavaScript que nos permiten declarar áreas de scroll, así como las barras de scroll asociadas a las mismas. En este caso, las clases concretas son `AppleScrollArea` y `AppleScrollbar` incluidas a partir de la versión 10.4.3 de Mac OS X.

Los pasos a seguir para utilizarlas son similares a los seguidos para el botón info:

- Incluir las clases en el documento HTML:

```
script type='text/javascript' src='AppleClasses/AppleScrollArea.js'
charset='utf-8'/>

<script type='text/javascript' src='AppleClasses/AppleScrollbar.js'
charset='utf-8'/>
```

- Introducir un elemento `<div>` para el contenido sobre el que hacer scroll y para la barra de scroll en el archivo HTML:

```
<div id="AreaScroll">...</div>

<div id="BarraScroll"></div>
```

- Declarar un controlador `onload`, invocado por JavaScript cuando se cargue el widget, donde se invoquen las constructoras de los objetos `AppleScrollArea`, `AppleScrollbar`, el código del controlador sería del estilo:

```
var gAreaScroll, gBarraScroll;

function setup() {

    gBarraScroll = new AppleVerticalScrollbar(document.getElementById
                ("BarraScroll"));

    gAreaScroll = new AppleScrollArea(document.getElementById
                ("AreaScroll"));


    gAreaScroll.addScrollbar(gBarraScroll);

}
```

Llamamos a las constructoras del área y la barra de scroll, pasándoles por parámetro el objeto del DOM donde se montarásn (las capas). Finalmente asociamos el área a la barra con el método `addScrollbar(gBarraScroll)`.

- Situar los elementos del contenido y la barra de scroll en sus capas correspondientes, haciendo uso de CSS para el formato
- Construir el área y barra de scroll mediante las clases de JavaScript `AppleScrollArea` y `AppleScrollbar`. Para tal efecto disponemos de un buen número de diferentes métodos y propiedades, que no incluiré puesto que son muy numerosas. Se pueden consultar en la documentación para desarrolladores de Apple.

Hay dos tipos de barras de scroll disponibles: verticales y horizontales, ambas subclases de la clase `AppleScrollbar`. Esta clase también incluye el material artístico (imágenes) necesario para la creación de las barras de scroll. Es posible sustituirlo por material propio, pero aquí usaremos el que viene por defecto. En el caso concreto que ahora nos ocupa, así como en los ejemplos, usamos las barras de scroll verticales.

Botones "Apple"

Apple proporciona una clase Javascript que simplifica la tarea de añadir botones de estilo predefinido en la interfaz de usuario de un widget. Esta clase se llama `AppleButton`, otra de las clases de Apple incluidas a partir de Mac OS X v10.4.3

La clase `AppleButton` incorpora todos los comportamientos estándar que esperaríamos de un botón. Para usar un botón de esta clase, hay que seguir los siguientes pasos:

- Incluir la clase `AppleButton` en el archivo HTML, de forma similar a las otras clases:

  ```
  <script type='text/javascript' src='AppleClasses/AppleButton.js'
  charset='utf-8'/>
  ```

- Representar el botón mediante un elemento de capa en el código HTML
- Declarar un controlador `onload`, una función JavaScript que se invoque cuando el widget se cargue que construya el objeto de la clase `AppleButton`
- Definir la posición del botón mediante CSS, de la misma manera que en el resto de objetos de clases
- Construir el botón a través de JavaScript usando la clase `AppleButton`.

Una vez el botón está creado, también se pueden modificar sus parámetros mediante JavaScript.

Una de las facetas más interesantes de cara a los desarrolladores de la clase `AppleButton`, es su subclase `AppleGlassButton`, que proporciona un estilo de botón estándar utilizado comúnmente en la parte trasera de los widgets. Este es nuestro caso de hecho, puesto que el botón que se utiliza en la parte posterior del widget pertenece a esta última clase. Por tanto, centraremos esta parte del anexo en explicar el funcionamiento de la subclase.

Un objeto de la clase AppleGlassButton tiene los siguientes atributos:

```
AppleGlassButton(

    buttonElement,
    label,
    onclick

);
```

Declararíamos la constructora del AppleGlassButton mediante un controlador del evento onload mediante JavaScript, usando un código de este estilo:

```
var gMyGlassButton;

function setup()

{

    gMyGlassButton = new AppleGlassButton(

        document.getElementById("miBotón"),

        "Haz Click",

        buttonClicked);

}
```

La variable `gMyGlassButton` almacena una referencia a la instancia del objeto `AppleGlassButton` creado, y nos permite interactuar con el botón. Estos son los métodos y propiedades disponibles:

OPCIÓN	TIPO	DESCRIPCIÓN
`gMyGlassButton.onclick`	Propiedad	Lectura/Escritura; controlador para el evento click
`gMyGlassButton.enabled`	Propiedad	Sólo lectura; retorna un booleano que especifica si el botón está activo o no
`gMyGlassButton.setEnabled(boolean)`	Método	Modifica el estado del botón.
`gMyGlassButton.remove()`	Método	Elimina el botón

Tabla 5.1: Propiedades y métodos del objeto `AppleGlassButton`

El objeto XMLHttpRequest

El uso de servicios web, cada vez más extendido, así como de manipulación de datos en formato XML, hace que en ocasiones sea necesario conectar un documento HTML directamente con datos XML para realizar actualizaciones en la página sin tener que volver a cargarla. AJAX (Asynchronous JavaScript And XML) facilita las tecnologías necesarias para tal fin. Entre ellas se encuentra el objeto `XMLHttpRequest`, que permite obtener y enviar información en formato XML. Los datos obtenidos se pueden procesar para convertirlos en contenido HTML válido. Esto se hace mediante el Modelo de Objetos del Documento, DOM (Document Objet Model), que lee el árbol de nodos del documentos y convierte la información en elementos HTML visibles para el usuario.

La sintaxis requerida para crear una instancia del objeto `XMLHttpRequest` difiere ligeramente entre los navegadores. En nuestro caso, nos basta con con invocar a la operación constructora del objeto de la siguiente manera:

```
var req = new XMLHttpRequest();
```

Esta operación retorna una referencia a un objeto abstracto. Sus métodos permiten controlar todas las diferentes operaciones, y sus propiedades contienen, entre otros campos, información retornada por el servidor.

Los métodos del objeto

Todas las instancias del objeto `XMLHttpRequest` , en todos sus diferentes entornos, comparte una lista de métodos y propiedades. La tabla muestra los métodos soportados en nuestro caso:

MÉTODO	DESCRIPCIÓN
`abort()`	Detiene la petición en curso
`getAllResponseHeaders()`	Retorna un conjunto de cabeceras en una cadena de texto
`getResponseHeader("headerLabel")`	Retorna la cadena de texto de una cabecera en concreto
`open("method", "URL"[, asyncFlag[, "userName"[, "password"]]])`	Asigna el método, la URL destino y otros atributos opcionales de una petición pendiente
`send(content)`	Transmite la petición, opcionalmente con una cadena de texto u objeto DOM
`setRequestHeader("label", "value")`	Asigna una pareja de valores a la cabecera enviada con la petición

Tabla 5.2: Métodos comunes del objeto `XMLHttpRequest`

De todos los métodos mostrados, los más destacables son `open()` y `send()`. El primero, `open()`, prepara el escenario para iniciar una operación. Solicita dos parámetros: el método HTTP que requerimos para la solicitud y la URL para establecer la conexión. El parámetro en forma de método, suele tomar los siguientes valores: "`GET`" para operaciones de obtención de datos y "`POST`" en operaciones para enviar información al servidor, especialmente si el tamaño de los datos enviados supera los 512 bytes. La URL por su parte puede ser completa o relativa.

El tercer parámetro es también importante. Es un booleano que determina si la transacción se controlará de forma asíncrona. El comportamiento por defecto, `true`, es el asíncrono, lo que quiere decir que el script que procesa los datos se ejecuta inmediatamente después de invocar el método `send`, sin esperar una respuesta. Por otra parte, si el valor del parámetros es `false`, el script espera hasta que la petición se envíe y se obtenga una respuesta desde el servidor. En nuestro caso, preferiremos un comportamiento asíncrono, puesto que en caso de que la red o el servidor tuvieran problemas, la transacción no se completaría nunca.

A continuación se muestra una función genérica que ilustra el funcionamiento del objeto. La función asume que la variable global `req` recibe el valor retornado por la constructora del objeto. El usar una variable global en este caso permite que los valores obtenidos puedan ser accedidos libremente desde cualquier otra función. También se da por sentada la existencia de una función `processReqChange()` encargada de controlar los cambios de estado de nuestro objeto:

```
     var req;

     function loadXMLDoc(url) {

          req = false;

               try {
                    req = new XMLHttpRequest();
                    }

               catch(e) {
                    req = false;
                    }

          if(req) {
               req.onreadystatechange = processReqChange;
               req.open("GET", url, true);
               req.send("");
          }
}
```

Esta función asigna a la variable global `req` una referencia al objeto creado. Si la creación ha tenido éxito, modificamos en primer lugar el parámetro onreadystatechange, asignándole un controlador que se invocará cada vez que se produzca un cambio de estado. Preparamos el escenario mediante un `req.open`, determinando en los parámetros que será una operación tipo "`GET`", obtención de datos, a la url especificada y de forma asíncrona. Se culmina la función con el método `send("")` para transmitir la petición.

Observación: Será esencial que los datos devueltos por el servidor sean enviado con `Content-Type` igual a `text/xml`. Si el contenido se envío como `text/plain` o `text/html`, la instancia del objeto también lo aceptará pero sólo se podrá utilizar mediante la propiedad `responseText`.

Las propiedades del objeto:

Una vez la petición ha sido enviada, los scripts pueden consultar diferentes propiedades, comunes a todas las diferentes implementaciones del objeto. Todas ellas son de lectura:

PROPIEDAD	DESCRIPCIÓN
`onreadystatechange`	Controlador para el evento invocado al cambiar de estado.
`readyState`	Un entero que representa el estado del objeto: 0 = no inicializado 1 = cargando 2 = cargado 3 = interactivo 4 = completo
`responseText`	Datos retornados por el servidor en cadena de texto
`responseXML`	Objeto de datos compatible con DOM retornados por el servidor
`status`	Código numérico devuelto por el servidor como 404 para "No encontrado" o 200 para "Ok"
`statusText`	Mensaje en forma de cadena de texto que acompaña el código de estado `status`

Tabla 5.3: Propiedades comunes del objeto `XMLHttpRequest`

La propiedad `readyState` se utiliza dentro del controlador que procesa los cambios de estado del objeto. Aunque el objeto puede pasar por diferentes estados durante su creación y procesado, el valor que señala el fin de la transacción es 4.

Antes de empezar a hacer operaciones con los resultados, es necesario confirmar el éxito de la transacción leyendo las propiedades status o `statusText`. Los valores 200 y OK respectivamente indican que todo ha ido bien.

El acceso a los datos devuelto por el servidor se hace mediante las propiedades `responseText` o `responseXML`. La primera consiste en una cadena de texto que contiene todos los datos. Es más interesante la segunda propiedades, que contiene un objeto documento XML. Este objeto es un DOM de tipo `nodeType` que se puede examinar y procesar mediante los métodos y propiedades de árboles y nodos del DOM que detalla el W3C. Esto permite encontrar elementos, atributos, etc. en el objeto.

A continuación se muestra la estructura básica de un controlador para la propiedad `onreadystatechange`, que autoriza el procesado de los datos sólo si todas las condiciones necesarias se cumplen.

```
function processReqChange() {

    // si el estado de req es “completado”
    if (req.readyState == 4) {
        // si “OK”
        if (req.status == 200) {
            // ...comandos para procesar la información...
        }
        else {
            alert("Se ha producido un problema al extraer los datos:\n" +
                req.statusText);
        }
    }
}
```

La función comprueba el estado del objeto. En caso de que la transacción se haya completado (`readyState = 4`) y se haya logrado éxito en la operación (`status = 200`), realizaremos las acciones pertinentes (e.g. procesar los datos obtenidos para mostrar los titulares de las noticias en el área de contenidos de nuestro widget). En caso contrario, avisamos al usuario de que no se ha podido realizar la consulta.